The Ballistic Research Laboratory CAD Package Release 4.0

VOLUME I

The BRL-CAD Philosophy

December 1991

Advanced Computing Systems
The U.S. Army Ballistic Research Laboratory
Aberdeen Proving Ground, Maryland 21005-5066

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
01-12-1991		

4. TITLE AND SUBTITLE
The Ballistic Research Laboratory CAD Package Release, 4.0, volume I: The BRL-CAD Philosophy

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Ballistic Research Laboratory
Aberdeen Proving Ground MD 21005-5066

8. PERFORMING ORGANIZATION REPORT NUMBER
BRL-CAD-R4-V1

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSORING/MONITORING AGENCY REPORT NUMBER

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The BRL-CAD Release 4.0 Manual is a collection of papers and manuals that gives an overview of BRL-CAD, explains installation and use of the package, and presents accomplishments and techniques performed using BRL-CAD. Papers included in this manual were written by members of the BRL-CAD development team as well as other users of the package. Many of the papers were submitted during the three BRL-CAD symposia held during 1988, 1989, and 1991. This manual in its entirety provides an extensive library of reference material for all types of users. Volume 1 contains articles written for symposia and magazines that discuss the development and purpose of BRLCAD.

15. SUBJECT TERMS
BOOLEAN EVALUATION ALGORITHM; TESSELLATORS; MGED; COMPUTER-AIDED-DESIGN, BRL-CAD(BALLISTICS RESEARCH LABORATORY COMPUTER AIDED DESIGN)

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
UNCLASSIFIED					Clifford Yapp
a. REPORT	b. ABSTRACT	c. THIS PAGE	U2	102	19b. TELEPONE NUMBER (*Include area code*)
U2	U2	U2			410-278-1382

Quality Assurance and OPSEC Review

This form is an approval record for ARL generated information to be presented or disseminated external to ARL.

CLICK TO REVIEW OR UPLOAD SOURCE MATERIALS: ARL Network | SAFE (Off Network)

FAQ

Section I - *To be completed by lead ARL author or COR*

A. General Information: ☐ Abstract ☐ BAA ☐ Invention Disclosure ☐ Multimedia ☐ Presentation ☐ Publication ☐ Report ☐ SBIR ☑ Other

1. Today's Date:	2. Due Date:	3. Unclassified Title or Solicitation Number/Title:
09/20/2013	10/11/2013	The Ballistic Research Laboratory CAD Package Release 4.0 - Volume 1 The BRL-CAD Philosophy

4. Author(s): (Last, First, MI):	5. Site & Office Symbol:	6. Telephone Number:
Muuss, Michael John; Dykstra, Phillip C.; Deitz, Paul H.; Mermagen, William H. Jr.; Stay, Paul R.; Reed, Harry L. Jr.; Davisson, Edwin O. Modern POC: Yapp, Clifford W.	APG RDRL-SLB-S	410-278-1382 7. Invited: ☐ Yes ☑ No

4b. Source Material Folder Location:	Y:\ARL_Form_1\SLAD\SLB-S\Yapp.C.20Sep13_BRLCAD4_Vol1
4c. Form 1 Author/POC's E-mail address:	clifford.w.yapp.civ@mail.mil
4d. Division Chief's E-mail address:	rachelle.m.moulsdale.civ@mail.mil
8. Contract No.:	
ARL COR:	
10. Distribution Statement: Is manuscript subject to export control? ☐ Yes ☑ No	9. Key Words:

11. Check Appropriate Letter and Number(s). See instructions:

A ☑ B ☐ C ☐ D ☐ E ☐ F ☐

0 ☑ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5 ☐ 6 ☐ 7 ☐ 8 ☐ 9 ☐ 10 ☐ 11 ☐ 12 ☐

12. Security Classification: Unclassified

B. Presentation ☐ Conference Presentation ☐ Notable Presentation ☐ Key Note Speaker ☐ Briefing

13. Conference / Meeting Name:	14. Conference / Meeting Location:

15. Conference / Meeting Date	16. Conference / Meeting is: ☐ Classified ☐ Other _____ ☐ Open to General Public ☐ Unclassified / Controlled Access

17. Will foreign nationals have access to this information? ☐ Yes ☐ No	18. Countries and International Agreement(s) of Foreign Nationals:

C. Publication ☐ Abstract ☐ Book ☐ Book Chapter ☐ Conference Proceeding ☐ Refereed Journal Article ☐ Other _____

19. Material will be submitted for publication in:

Journal Title: Country:

D. Report ☐ Contractor Report ☐ Memorandum Report ☐ Reprint ☐ Special Report ☐ Technical Note ☐ Technical Report

20. Project No.:	21. Period Covered (mm/yyyy)	22. Sponsoring Agency:
	12/1991	Ballistic Research Laboratory

E. Multimedia ☐ Software ☑ Web ☐ Poster ☐ Video Clip ☐ Other _____

23. Location: DTIC, ARL website, BRL-CAD website, etc.

F. Author's Statement

24. All authors have concurred in the technical content and the sequence of authors. All authors have made a substantial contribution to the manuscript, and all authors who have made a substantial contribution are identified in Block 4.

ARL Lead Author or COR X YAPP.CLIFFORD.WILLIAM.1364725708 Digitally signed by YAPP.CLIFFORD.WILLIAM.1364725708 DN: cn=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=YAPP.CLIFFORD.WILLIAM.1364725708 Date: 2013.09.20 11:44:20 -04'00' Date 09/20/2013

Section II - OPSEC Review Checklist *To be completed by an ARL Trained Internal OPSEC Reviewer*

OPSEC POC: Complete and explain any positive responses in block 28.
Note: ARL must be the proponent of the proposed information for release.

1. Does this material contain Sensitive Information?	☐ YES ☑ NO	7. Does this material contain:	
2. Does this information contain state-of-the-art, breakthrough technology?	☐ YES ☑ NO	a. Any contract proposals, bids, and/or proprietary information?	☐ YES ☑ NO
3. Does the United States hold a significant lead time in this technology?	☐ YES ☑ NO	b. Any information on inventions/patent application for which patent secrecy orders have been issued?	☐ YES ☑ NO
4. Does this information reveal aspects of reverse engineering?	☐ YES ☑ NO	c. Any weapon systems/component test results?	☐ YES ☑ NO
5. Does this material reveal any security practices or procedures?	☐ YES ☑ NO	d. Any ARL-originated studies or after action reports containing advice and recommendations?	☐ YES ☑ NO
6. Would release of this information be of economic benefit to a foreign entity, adversary, or allow for the development of countermeasures to the system or technology?	☐ YES ☑ NO	e. Weakness and/or vulnerability information?	☐ YES ☑ NO
		f. Any information on countermeasures?	☐ YES ☑ NO
		g. Any fielding/test schedule information?	☐ YES ☑ NO

Section II - OPSEC Review Checklist Continued

h. Any Force Protection, Homeland Defense (security) information?	☐ YES	☑ NO	k. Contain information with foreign policy or foreign relations implications?	☐ YES	☑ NO
i. Information on subjects of potential controversy among military services or other federal agencies?	☐ YES	☑ NO	l. Does this information contain current ARL Essential Elements of Friendly Information (EEFI)?	☐ YES	☑ NO
j Information on military applications in space, nuclear chemical or biological efforts: high energy laser information; particle beam technology; etc?	☐ YES	☑ NO			

G. Internal OPSEC Review

25. OPSEC Review Comments / Explanations / Continuations:

26. Internal OPSEC Approval Statement: N/A: ☐

I, the undersigned, am aware of the adversary's interest in DoD publications and in the subject matter of this material and that, to the best of my knowledge, the net benefit of this release out weighs the potential damage to the essential security of all ARL, AMC, Army, or other DoD programs of which I am aware.

Penny T. Willard

X WILLARD.PENNY.T.1282391488 Digitally signed by WILLARD.PENNY.T.1282391488 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=WILLARD.PENNY.T.1282391488 Date: 2013.09.20 14:44:35 -04'00'

OPSEC Reviewer Name — OPSEC Reviewer Signature — Date 09/20/2013

Section III - *Approval to be completed by designated individual(s)*

H. Technical Review

27. Technical Reviewer Name(s)	Technical Reviewer Signature(s)	Date(s)
	X BOWMAN.WILLIAM.KEITH.13805723 Digitally signed by BOWMAN.WILLIAM.KEITH.13805723 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=BOWMAN.WILLIAM.KEITH.13805723 Date: 2013.09.23 10:22:06 -04'00' 34	
	X	
	X	

I. Supervisor Review

28. Supervisor Name — X PERRY.MARTIN.J.1043561320 Digitally signed by PERRY.MARTIN.J.1043561320 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=PERRY.MARTIN.J.1043561320 Date: 2013.09.23 15:25:07 -04'00' Supervisor Signature — Date 09/23/2013

J. Technical Publications Editorial Review

N/A: ☐

29. Editor Name — X Editor Signature — Date

Section IV - *To be completed by designated individual*

K. Security

30. ARL OPSEC Officer *Material has been reviewed for OPSEC Policy and* **IS** ☑ / ☐ **IS NOT** *acceptable for public release.*

Comments:

ARL OPSEC Officer Name Daniel B Cramer — X CRAMER.DANIEL.BRUCE.1229367800 Digitally signed by CRAMER.DANIEL.BRUCE.1229367800 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=CRAMER.DANIEL.BRUCE.1229367800 Date: 2013.09.24 06:38:42 -04'00' 9367800 ARL OPSEC Officer Signature — Date 09/24/2013

31. Classified Information: N/A: ☐

Classified by/Derived from: _____

Declassified on date: — Security Manager X — Date

32. Foreign Intelligence Office *(Limited distribution information)* N/A: ☐

FIO Reviewer Name — X FIO Reviewer Signature — Date

33. Foreign Disclosure Office N/A: ☐

(Limited distribution information for release to foreign nationals)

FDO Reviewer Name — X FDO Reviewer Signature — Date

L. Division Chief

34. The information contained in this material has received appropriate technical / editorial review and ☑ **IS** / ☐ **IS NOT** *acceptable for public release.*

Shelly Moulsdale, for Bob Bowen

Division Chief Name — X MOULSDALE.RACHELLE.M.1255778304 Digitally signed by MOULSDALE.RACHELLE.M.1255778304 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=MOULSDALE.RACHELLE.M.1255778304 Date: 2013.09.26 06:55:56 -04'00' 78354 Division Chief Signature — Date 09/26/2013

M. Public Affairs Office

35. The information contained in this material has received appropriate technical / editorial review and ☑ **IS** / ☐ **IS NOT** *approved for public release.*

Joyce M. Conant

PAO Reviewer Name — X CONANT.JOYCE.M.1161860308 Digitally signed by CONANT.JOYCE.M.1161860308 DN: c=US, o=U.S. Government, ou=DoD, ou=PKI, ou=USA, cn=CONANT.JOYCE.M.1161860308 Date: 2013.09.26 07:27:55 -04'00' 61860308 PAO Reviewer Signature — Date 09/26/2013 N/A: ☐

VOLUME I - THE BRL-CAD PHILOSOPHY

ABOUT BRL-CAD

The BRL-CAD package combines a powerful solid-modeling capability with network-distributed image processing. This software is currently installed and running at over 800 sites. BRL-CAD started in 1979 as a task to provide an interactive graphics editor for the BRL vehicle-description database. Today, the package with over 100 programs totals more than 280,000 lines of "C" source code, runs under the UNIX operating system, and is supported by more than a dozen product lines, from Sun Workstations to the Cray 2. The BRL-CAD package includes four major features:

- A solid geometric editor
- The ray tracing library
- Two lighting models
- Many image-handling, data-comparison, and other support utilities.

Release 4.0 of the BRL-CAD package also supports several geometrical representations of data:

- The original Constructive Solid Geometry (CSG) BRL database
- Extensions that include solids made from collections of Uniform B-Spline surfaces and Non-Uniform Rational B-Spline (NURB) surfaces
- A faceted data representation
- n-Manifold Geometry (NMG).

THE RELEASE 4.0 MANUAL

The BRL-CAD Release 4.0 Manual is a collection of papers and manuals that gives an overview of BRL-CAD, explains installation and use of the package, and presents accomplishments and techniques performed using BRL-CAD. Papers included in this manual were written by members of the BRL-CAD development team as well as other users of the package. Many of the papers were submitted during the three BRL-CAD symposia held during 1988, 1989, and 1991. This manual in its entirety provides an extensive library of reference material for all types of users.

Because the material covered by the Release 4.0 Manual is extensive, the manual has been divided into five volumes and is organized to help readers find the information needed. The volumes are also bound separately for ease of use. BRL-CAD users who are interested in specific subjects can find the appropriate

articles in either the Table of Contents for each volume or the Permuted Index. Brief descriptions of the material found in each volume follow:

VOLUME I - The BRL-CAD Philosophy
This volume contains articles written for symposia and magazines that discuss the development and purpose of BRL-CAD.

VOLUME II - The BRL-CAD User's Manual
This volume contains instructions for installing BRL-CAD, the man pages for the commands, libraries, and file types, and short documents pertaining to the updates in Release 4.0 as well as short documents previously included in Release 3.0.

VOLUME III - The BRL-CAD Applications
This volume contains supplemental applications written and discussed by various users of BRL-CAD. Some sections are mini-manuals that explain commands in more detail than the man pages in Volume II.

VOLUME IV - The MGED User's Manual
This volume contains the stand-alone user's manual written for MGED as a guide for constructing and editing BRL-CAD solid models.

VOLUME V - The BRL-CAD Analyst's Manual
This volume consists of papers written for the 1988, 1989, and 1991 BRL-CAD symposia and memorandum reports discussing the usefulness and progress of the software. The articles are arranged according to subject matter, so an individual interested in raytracing, for example, can flip to the papers that discuss this feature. The permuted index at the end of this volume directs the reader to articles based on keywords or phrases. The permuted index lists keywords that are referenced in articles, regardless of the paper's subject matter. Using this index, one can quickly locate papers containing the topic of interest.

HOW TO GET STARTED

It is suggested that users who are new to BRL-CAD first review Volume I. The overview papers explain solid modeling, surface modeling, and hybrids and describe the techniques employed by BRL-CAD. The article *A Road Map Through the BRL-CAD Package* gives an excellent explanation of the data flow and approach to using BRL-CAD. Other articles also describe the history and development of the package and new features in Release 4.0.

When it is time to install the package, *Installing the BRL-CAD Package, Release 4.0* in Volume II explains the installation procedures and the configuration required. The man pages included in the next section of the volume explain all the commands available, the libraries used by the package, and the appropriate file formats. Volume II also provides a handy reference for users who are editing, interrogating, or analyzing the models. Short documents and E-mail discussions that occurred during development and publication of Release 4.0 follow the man pages. Release 3.0 Notes and Errata Sheets are found in the last sections of this volume. Papers discussing supplemental commands and applications to BRL-CAD are included in Volume III.

For users interested in creating and editing solid models, Volume IV explains the Multiple-Device Graphics Editor (MGED) in detail. It is in the format of a stand-alone user's manual and provides instructions about solid modeling, available commands, and useful techniques. This manual should be kept within easy reach of the graphics workstation.

Those who are interested in the various uses for BRL-CAD, efforts accomplished using the package, or techniques for using BRL-CAD as an analysis tool should refer to Volume V. This volume comprises numerous papers written for the three BRL-CAD symposia as well as separately-published articles. For convenience, this manual is divided into sections, each representing a subject contained in the articles, such as benchmarking, modeling, radar, survivability/vulnerability, etc. One paper may reference several subjects, but in each case, the overall or major subject was chosen for organizational purposes. Refer to the Table of Contents located at the beginning of the volume for a list of the subjects and the articles contained within the subject of interest.

An index has also been provided in Volume V to help readers find the articles referring to topics of interest. The index is permuted; that is, it lists keywords found in the abstracts and introductions of the articles and for every key word lists the titles of the articles in which it was found. A reader interested in a particular topic can look up a single word or phrase and find all the articles that use those words. For example, suppose the reader is researching how BRL-CAD was used for radar models. A quick check against the permuted index shows the word "radar" used in 13 different articles. The Table of Contents for Volume V only lists five articles with radar as the major subject. For more information about radar, the user may refer to the other eight articles indicated by the permuted index.

VOLUME I - THE BRL-CAD PHILOSOPHY

TABLE OF CONTENTS

BRL-CAD RELEASE 4.0
FEATURES AND DISTRIBUTION INFORMATION

The BRL-CAD Package is a powerful Constructive Solid Geometry (CSG) computer based solid modeling system. BRL-CAD includes an interactive geometry editor, a ray tracing library, two ray-tracing based lighting models, a generic framebuffer library, a network-distributed image-processing and signal-processing capability, and a large collection of related tools and utilities. Release 4.0 is the latest version of software which has been undergoing continuous development since 1979.

The most significant new feature for Release 4.0 is the addition of n-Manifold Geometry (NMG) support based on the work of Kevin Weiler. The NMG software converts CSG solid models into approximate polygonalized boundary representations suitable for processing by subsequent applications and high-speed hardware display.

BRL-CAD is used at over 800 sites located throughout the world. It is provided in source code form only, and totals more than 280,000 lines of "C" code.

BRL-CAD supports a great variety of geometric representations, including an extensive set of traditional CSG primitive solids such as blocks, cones and torii, solids made from closed collections of Uniform B-Spline Surfaces as well as Non-Uniform Rational B-Spline (NURBS) Surfaces, purely faceted geometry, and n-Manifold Geometry (NMG). All geometric objects may be combined using boolean set-theory operations such as union, intersection, and subtraction.

Material properties and other attribute properties can be associated with geometry objects. Combining material properties with geometry is a critical part of the link to applications codes. BRL-CAD supports a rich object-oriented set of extensible interfaces through which geometry and attribute data are passed to applications.

A few of the applications linked to BRL-CAD include:

- Optical Image Generation (including specular/diffuse reflection, refraction, multiple light sources, and articulated animation)
- An array of military vehicle design and evaluation Vulnerability/Lethality (V/L) Codes
- Bistatic laser analysis
- Predictive Synthetic Aperture Radar Codes (including codes due to ERIM)
- High-Energy Laser Damage
- High-Power Microwave Damage
- Weights and Moments-of-Inertia
- Neutron Transport Code
- PATRAN™, ADINA, EPIC-2, NASTRAN, etc. for structural/stress analysis
- X-Ray image calculation

BRL-CAD requires the UNIX operating system and is supported on more than a dozen product lines from workstations to supercomputers, including: Alliant FX/8 and FX/80, Alliant FX/2800, Apple Macintosh II, Convex C1, Cray-1, Cray X-MP, Cray Y-MP, Cray-2, Digital Equipment VAX, Gould/Encore PN

6000/9000, IBM RS/6000, Pyramid 9820, Silicon Graphics 3030, Silicon Graphics 4D "Iris", Sun Microsystems Sun-3, and the Sun Microsystems Sun-4 "SparcStation". Porting to other UNIX systems is very easy, and generally only takes a day or two.

You may obtain a copy of the BRL-CAD Package and distribution materials in one of two ways:

1. FREE distribution with no support privileges: Those users with online access to the DARPA InterNet may obtain the BRL-CAD Package via FTP file transfer, at no cost, after completing and returning a signed copy of the enclosed agreement and survey form. There are encrypted FTP-able files in several countries around the world. Directions on how to obtain and decrypt the files will be sent to you upon receipt of your signed agreement. One printed set of BRL-CAD documentation will be mailed to you at no cost. Note that installation assistance or telephone support are available only with full service distributions. Upgrade to full service status can be made at any time by following instructions under 2 below. Please send the signed distribution agreement and survey form via ISO Group III FAX to USA (410) 278-5058 or mail to:

BRL-CAD Distribution
ATTN: SCLBR-LV-V
Aberdeen Proving Ground, MD 21005-5066
USA

For further details, E-mail to <keith@brl.mil>, send message to number above or write to the above address.

2. FULL SERVICE distribution: The Survivability/Vulnerability Information Analysis Center (SURVIAC) administers the supported BRL-CAD distributions and the information exchange programs for BRL through the SURVIAC Aberdeen Satellite Office. The full service distribution cost is US $500 (no cost to US Government Agencies). A copy of the BRL-CAD package will be provided to you on your choice of magnetic tape media. You may also elect to obtain your copy via network FTP. One printed set of BRL-CAD documentation will also be mailed to you. BRL-CAD maintenance releases and errata sheets will be provided at no additional charge, and you will have access to full technical assistance by phone, FAX, letter, or E-mail. Complete and return a signed copy of the enclosed distribution agreement and survey form with a check or purchase order payable to "BA&H/SURVIAC" and mail to:

BRL-CAD Distribution
SURVIAC Aberdeen Satellite Office
1003 Old Philadelphia Road
Suite 103
Aberdeen, MD 21001 USA

For further details, call Ms. Carla Moyer at USA (410)-273-7794, send E-mail to <cad-dist@brl.mil>, send ISO Group III FAX message to USA (410)-272-6763, or write to the above address.

All users have access to the BRL-CAD Symposia, workshops, user's group, and BRL-CAD information via E-mail.

BRL-CAD:
Recent Progress
and
Future Directions

Michael John Muuss

Ballistic Research Laboratory
Aberdeen Proving Ground
MD 21005-5066 USA

ABSTRACT

The current production release of the BRL-CAD Package is Release 3.7, which has been in use since June of 1989. In the intervening years, a lot of work has been accomplished, and these new developments will be made available as Release 4.0. A brief review of the improvements that have been added in Release 4.0 will be given.

Many projects are planned for the future, or are in an early stage of investigation. Some of these projects are a natural continuation of the work of the past few years, while other projects represent bold departures into new areas for BRL-CAD. To give BRL-CAD Package users an idea of what developments can be expected, this paper touchs on the highlights of future work.

April 14, 1991

BRL-CAD:
Recent Progress
and
Future Directions

Michael John Muuss

Ballistic Research Laboratory
Aberdeen Proving Ground
MD 21005-5066 USA

The BRL-CAD Package as it stood in 1989 was one of the few production CAD systems which was based purely on the combinatorial solid geometry (CSG) technique. The final shapes of all objects were created by boolean combinations of primitive solids. No attempt was made to represent either the topology or the surface geometry of any object in explicit form. The exact nature of the final shape of an object was discovered only on a point by point basis by sampling the object with ray-tracing. At the same time, the assortment of primitive solids available to designers was a hybrid of traditional CSG primitives and more recent boundary-representation (B-rep) primitives. The primitive solids described by their boundaries included a variety of faceted solids, as well as solids defined by a closed collection of non-rational B-spline surfaces. This rich collection of primitive solids can be combined by the designer through any number of boolean operations. However, complex combinations of primitives can be difficult to visualize except through the careful study of their three dimensional wireframes rotating in real time coupled with the judicious use of ray-traced renderings.

1. Improvements added in Release 4.0

1.1. n-Manifold Geometry

Given that modern computer workstations with integral polygon rendering hardware [Molnar87a] are now commonly available, and that hardware rendering speeds can exceed one million polygons per second, it seems highly desirable to take advantage of this hardware. Also, there is a wealth of analysis software that simply can not make do with a geometry interrogation interface that supports only the ray-tracing paradigm. What was missing from the BRL-CAD Package was a way of obtaining an *explicit* description of the final shape of modeled objects.

The single most significant BRL-CAD effort in recent times has been the addition of the n-Manifold Geometry (NMG) support. The NMG data structures have the advantages of complete generality and closure under boolean operations [Muuss90a]. NMGs also encode the full topological structure of an object, as well as the geometric information. This representation contains full topology information, so that the relationships between vertices, edges, loops, faces, and shells are continuously available. Geometry is associated with each topological element. Because of the frequency of occurrence of non-3-manifold conditions in CSG modeling, both intentionally and as part of various analysis operations, the NMG data structures were essential.

The addition of NMG support to the BRL-CAD Package is concentrated in several key areas. The existing combinatorial solid geometry database remains unchanged. Thus, any application that needs an explicit representation of the model obtains that explicit representation by converting the CSG database with a "post-processing" operation. Much more detail is available in [Muuss91a].

1.1.1. The NMG Library Routines

The applications programmer is insulated from the details of the NMG radial-edge data structures by a library of functions which perform all of the basic tasks. Each of these routines accepts a valid NMG model or a part of one, and performs an operation, returning a valid model upon completion. The routines within the library are organized into three groups: constructive, destructive, and manipulative.

1.1.2. Tolerance-Driven Tessellators

The job of a tessellator is to convert a given solid primitive into a faceted approximation stored in NMG data structures. There are two aspects to this conversion: establishing the topology of the approximation, and then generating the geometry to associate with the approximate topology.

Conversion of curved, implicitly defined primitive solids to a faceted representation will necessarily be inexact. Three types of tolerances are passed to the tessellator to provide control over the nature and magnitude of the errors introduced by the faceted approximation used in the tessellation. The *absolute* tolerance, which limits the maximum permissible difference between any point on the tessellation and the corresponding point on the original solid, is expressed as an absolute distance. For example, using this mechanism it is possible to ensure that no face deviates from the true surface by more than 2 mm. The *relative* tolerance also limits the maximum error of any point, but it is expressed as a fraction (between 0.0 and 1.0) of the diameter of the bounding sphere which encloses the original solid. Finally, the *normal* tolerance limits the maximum angular error of the surface normal. This normal tolerance permits users to make statements about the accuracy of the surface normals.

1.1.3. The Boolean Evaluation Algorithm

As each boolean operation is encountered in the database, it is performed on the appropriate tessellated solids. *nmg_do_bool()* takes the two tessellated objects and combines them according to the boolean operation, resulting in a consistent set of solid tessellated objects [Muuss90a]. Once all the objects have been retrieved from the database and combined with their boolean formulaes, the resultant collection of NMG objects is supplied to the application.

1.1.4. Extensions to MGED

The majority of a BRL-CAD user's contact with the new non-manifold geometry capability will be through **mged** [Applin88a]. The user can request the calculation of approximate boundary wireframes. The evaluation is performed by tessellating each of the primitive solids into an NMG object meeting the current tolerance, and combining them according to the indicated boolean operations. This operation is invoked with the command

mged> *ev -w object*

Note that the evaluated boundary wireframes are not stored in the database and are primarily intended as a visualization aid for the designer.

On those hardware platforms where polygon drawing capabilities exist, it is possible to have a flat-shaded polygonal rendering of database objects drawn. This operation is invoked with the *evaluate* command

mged> *ev object*

Once the polygonalized version of the object is on the screen, it can be rotated in real time. This capability gives the designer the opportunity to more fully appreciate the complex shape which has been created, and to judge whether the evaluated shape matches the intended design.

On hardware platforms, such as the Silicon Graphics 4D workstations, which have hardware support for drawing shaded polygons lit with multiple light sources, it is possible to activate the hardware lighting model. This provides a much more realistic rendition of the evaluated objects.

1.1.5. Export of Polygons to Other Applications

The ability to produce an explicit approximate representation of the surface of any object stored in the geometry database exists as a general capability. Any application program that links to the library **librt** can use this capability at any time in the analysis process. Furthermore, use of this capability can be simultaneously intermixed with other forms of interrogation supported by the library, so that an application can perform some operations using the approximate surface description and other operations using ray-tracing. The application has no knowledge of the underlying primitives used to describe the objects in the database, nor is the application aware that the library creates the surface description by extracting the objects from the geometry database, tessellating them into NMG solids, and then combining them via boolean formulas.

There may be circumstances when a designer wishes to take a collection of primitive solids, tessellate and combine them into some faceted shape, and then store the faceted shape as the finished design. This can be accomplished with the **mged** *facetize* command:

mged> *facetize newsol oldsol*
mged> *facetize newreg oldreg*

The *facetize* command takes either a single pre-existing solid or a single pre-existing combination of solids (such as a group or region), tessellates them according to the current tolerance settings, and creates a new facetized solid in the database. This resulting solid is represented using the NMG data structures, and has exactly the same standing as any other primitive solid: it can be ray-traced, instanced, and combined with other solids to create new shapes. However, no history is kept about the formation of the new NMG solid. Thus, changes made to the parameters of an original solid inside the original region *oldreg* are *not* propagated to the facetized version *newreg*.

1.2. Support of mged on the SGI

SGI lighting model support in **mged** is for lit polygons. A very preliminary version of lighting model support in **mged** exists to help debug the NMG tessellation routines. It uses four infinite lights of dubious color and location, to help see the facets (eg, on the TGC) very clearly. As usual, getting the SGI to do anything useful was a much bigger fight than expected. A project for the future is to program the lighting model on the SGI to use the same Phong parameters that **rt** uses when rendering each object.

mged was extended to retain the color information found when reading UNIX-plot files with the *overlay* command This has proved helpful for examining some of the NMG debugging plot files. It is completely general, allowing any color UNIX-Plot file to be used as an overlay. Since in **mged** each solid can be drawn in exactly one color, this is accomplished by accumulating a different _PLOT_OVERLAY "phantom" solid for vectors of each color.

- It is now possible to move and resize the **mged** window. In previous releases, resizing the window did not work. It seems that the SGI write-protects all new bits added to a window on resize or move until they are made writable with *scrmask()*.
- Accounts for 4:3 aspect in NTSC mode, so that circles are round on TV.
- Clears the full viewport in NTSC mode, to clean out other window stuff that popped up in the interim, such as a **libfb** window created by running **rt** from within **mged**.

1.3. New Spline Library

The old spline library in Release 3.7 has been replaced by a completely new spline library. The new library is able to interpolate *n*-dimensional data: e.g. X,Y,Z spatial dimensions plus R,G,B and temperature, simultaneously across the same surface. This library also facilitates new modeling operations: surfaces of revolution, and general extrusions. In addition to modeling applications, this n-dimensional technique is useful for applying a camouflage pattern to the surface, mapping thermal data onto the surface, and other energy/geometry mappings.

librt support for ray-tracing NURBS solids has been improved by the addition of much

more robust ray intersections. This solves the "crack" problem which happened frequently when the old algorithm subdivided the spline into polygons. Because different parts of the spline needed different amounts of subdivision, edges of adjoining polygons did not always match up. The new algorithm does not generate polygons as an intermediate step. This method uses much less memory while still converging to a solution of the ray/spline intersection in similar time. The new approach is a fully parallel algorithm, with no interlocking or critical sections. On parallel CPUs, this eliminates the serious bottleneck of the previous algorithm.

The new NURBS Algorithm first converts the NURBS surfaces into Bezier patches, and then transforms each Bezier patch into a projective (2-D) space, where the ray is located at origin of the projection. This reduces the problem from having to compute with 5-tuples to computing with only 2-tuple, resulting in 60% less computation for every intersection. Bezier clipping is used to locate origin, by finding the zeros of the polynomial.

1.4. New Primitive Solids

Improved geometric coverage is the goal. In addition to adding a variety of new traditional (GIFT) primitives, NMG objects may also be stored in the database as primitives. The traditional primitives that have been added are the arbitrary convex faceted solid (ARBN), the halfspace (HAF), and wire solids (WIR). The wire solids are implemented in the much more general form of a "pipe" solid, which is either solid or hollow. A pipe solid is composed of an arbitrary number of segments, where each segment is either linear with constant radius, linear with changing radius, or bending with constant radius. In addition, the "particle" solid was added to support the use of particle systems. Addition of new ERIM solids is planned for the near future.

1.5. COMGEOM-G: GIFT Database Importer

This is a new COMGEOM "card deck" to BRL-CAD geometry database converter. This function was formerly provided by the (unsupported) programs **cvt4** and **cvt5**. **comgeom-g** can handle both "current" GIFT versions (version 4 and version 5) of the input "deck" format, as well as the traditional (version 1) MAGIC format, which is still used by the Department of Energy. Automatic units conversion is built-in. Furthermore, it properly converts the halfspace (HAF), ARBN, and WIR solids, which previous converters did not do. **comgeom-g** writes the **mged** database by employing **libwdb** subroutines.

1.6. G2ASC, ASC2G: ASCII Database Converters

Support of the particle and pipe solids has been added, as well as support of ARBN solids. The new **asc2g** uses **libwdb** for most geometry output operations, making it smaller and much easier to understand and maintain. This is also an important precursor to the planned (post-Release-4.0) database format change.

1.7. More image/signal processing tools

Many of Phil Dykstra's private stock of image processing tools have been polished up and included in the distribution. This includes **libfft**, a library for evaluating FFTs, and a new directory ("sig") of tools for signal processing, including a variety of windowing and filtering programs.

1.8. Improvements to LIBFB

The framebuffer server **fbserv** is the most prominent addition to **libfb**. **fbserv** provides the bridge between the framebuffer semantics of **libfb** and the transient nature of the windowing environment found on most modern graphics workstations [Muuss89a]. An arbitrary number of **fbserv** processes can be run, with each one being given a different identifying number (usually small integers like 0, 1, 2). Each **fbserv** opens a window on the workstation, and then blocks, awaiting an incoming **libpkg** connection from a **libfb** graphics display program. Between uses, the **fbserv** process continues to exist, maintaining the state of the window for the next application.

More device support was added to **libfb**, including improvements to the SGI interface, the addition of support for the Abekas A60 digital videodisk, and preliminary support for the Tek-source framebuffer.

In Release 4.0, there has been a slight shift in the philosophy of the library. **libfb** now has to be linked with the system-specific libraries indicated by the **cake** configuration variable LIBFB_LIBES, because copies of all the required vendor libraries are no longer included within the **libfb** archive. Thus, **libfb** is no longer "self-contained", but may continue to reference routines from system-specific libraries. This has the benefit of greatly reducing the size of **libfb**. For example, the copy of **libfb** for the SGI made the old way was truly enormous, because it included complete copies of **libgl**, **libX11**, and **libnet**, as well as the actual **libfb** routines. This new strategy permits more flexibility in working around difficulties, such as using X11 libraries created by BLD on the Crays.

This change forces **libfb** applications to use Cakefiles, or have Makefiles that are hand-adjusted on a per-system basis. Instead, Cakefile.defs defines a new symbol, LIBFB_LIBES, that lists any and all loader options that need to be used when linking against **libfb**. Thus, all Cakefile rules should look like:

LIB_PRE"LIBFB LIBFB_LIBES

to refer to the framebuffer library. All the distributed Cakefiles were modified to reflect this change.

Some new subroutine interfaces have been added to the library. *fb_view()* and *fb_getview()* replace the viewport, window, zoom routines. *fb_getcursor()* and *fb_poll()* have also been added.

libfb/if_disk.c was extended so that the framebuffer named "—" (single minus) is now a synonym for standard output (**stdout**). If a program that writes to a framebuffer is "well-behaved" and writes scanlines in ascending order, then that program can be used in a pipeline for further processing. As a simple but potent example,

fbclear —F— 100 200 100 | pixmatte ....

is a way of creating a file of a given color of exactly the right size. (A better way of actually accomplishing this task is with **gencolor**). For programs that are not well-behaved in writing to the framebuffer, **stdout** can be stacked with the /dev/mem interface, meaning that commands like

fbgrid -F "/dev/mem —" | pixmatte ....

can also be used in this way.

The **fbhelp** program will return information about the framebuffer choices currently available, and details on the current selection. Here is an example output:

```
A Frame Buffer display device is selected by
setting the environment variable FB_FILE:
(/bin/sh ) FB_FILE=/dev/device; export FB_FILE
(/bin/csh) setenv FB_FILE /dev/device
Many programs also accept a "-F framebuffer" flag.
Type "man brlcad" for more information.
================ Available Devices ================
/dev/sgi      Silicon Graphics Iris '4D'
/dev/debug    Debugging Interface
/dev/ab       Abekas A60 Videodisk, via Ethernet
/dev/stack    Multiple Device Stacker
/dev/mem      Memory Buffer
/dev/null     Null Device
host:[dev]    Remote Device Interface
filename      Disk File Interface
================ Current Selection ================
```

Description: Silicon Graphics Iris '4D'
Device: /dev/sgi
Max width height: 1280 1024
Default width height: 512 512
Usage: /dev/sgi[options]
p Private memory - else shared
l Lingering window - else transient
f Full centered screen - else windowed
t Thirty Hz (e.g. Dunn) - else 60 Hz
n NTSC+GENLOCK - else normal video
e External sync - else internal
c Perform software colormap - else use hardware colormap on whole screen
G Don't use GT & Z-buffer hardware, if present (debug)
s On GT, single buffer, don't double buffer
z Zap (free) shared memory

This program can be a big help in determining what framebuffer options have been selected, and what other choices may be available. It describes remote and stacked collections of framebuffers, as well as simple local framebuffer references.

1.9. Improvements to librt

librt was extended to include a general collection of point, line, and plane subroutines, such as *rt_mk_plane_3pts()*, *rt_mkpoint_3planes()*, and *rt_isect_ray_plane()*.

The object-oriented interface between **librt** and the individual geometry modules has been extended, with the addition of several new operations that each geometry module is required to implement. First, a distinction is now made between the format of a primitive solid when it is located in the geometry database on disk, and when it is in memory, intended for processing by a program. A future extension to the **mged** database format will make the database become machine independent, while the in-memory representation must be in the hardware's native internal format. As the first step in this direction, all BRL-CAD code is being reorganized so that only the **librt** geometry modules have knowledge of the on-disk format of solids. Through the geometry modules *ft_import()* and *ft_export()* interfaces, the opaque format disk information can be converted into an internal hardware specific format. The internal format is now described by the header file "h/rtgeom.h". The internal format can be freed with *ft_ifree()*, and the solid that it represents can be described by *ft_describe()*. The description is returned in a variable length string (**struct rt_vls**), as described by the header file "h/rtstring.h". The *ft_tessellate()* interface causes the solid to be approximated as a collection of planar faces, in an NMG **nmgregion** structure.

In Release 3.7, the **rt_vlist** structures used within **mged** and **librt** to represent "display list" contained a single point and command per data structure. Each data structure was individually allocated with *malloc()*. A given solid ranges from having dozens to tens of thousands of points in its display list; with the old t_vlist structures, this resulted in an inefficient use of memory and a lot of needless time spent in *malloc()*. New "chunky" **rt_vlist** structures have been instituted, and the **librt**, **rt**, and **mged** directories have all been converted. By grouping RT_VLIST_CHUNK (currently 35) points together, much more efficient use of memory is made. On 32 bit machines, each **rt_vlist** structure weighs in at just under 1k bytes, making it a convenient size to manage.

Phil Dykstra's Quaternion math package [Shoemake85a] was added to **librt** in the file "librt/qmath.h". It has also been supplemented with many of the quaternion calculus operations such as natural logarithm (ln), exponentiation (exp), spherical linear interpolation (stable slerp), etc [Shoemake89a]. The macros have been added to h/vmath.h, and the subroutines now live in librt/qmath.c.

The robustness of the root finder package in **librt** was greatly improved. This resolved a bug with certain TGC solids when viewed from an azimuth of zero degrees, and overall should

help produce better results.

In Release 3.7, every time a program needed to traverse the directed acyclic graph (DAG) of a database tree, a small piece of specialized code was written. This strategy resulted in the proliferation of "tree walker" routines. The addition of the NMG *ev* command to **mged** required another tree walker a little different from all the existing ones. This motivated the creation of one fully general tree walker, capable of replacing all of the half-dozen existing ones. As a result of this, all the features described below apply uniformly to **rt**, **lgt**, etc., as well as to all the viewing commands in **mged** (*e*, *ev*, *B*, etc.).

Non-union operations are now supported in combination nodes which are located above region nodes. To get a cutaway view of a whole model, it used to be necessary to individually remove the cutout solids from each and every region in the model. Now, for example, a top-level group can now be created which is "vehicle minus cutout". This vastly simplifies the construction of "molds" for parts, and for making cutaway views for improved design visualization.

When naming objects in the database, the traditional approach of naming just the treetop can be used, or partial or full path specifications can be given, with any modeling transformations accumulated along the way. For example, within **mged** it is now possible to give the command

```
e vehicle.g/suspension/l-track/idler  vehicle.g/engine/drive-train
```

and see the "idler" and "drive-train" assemblies drawn in their final locations, relative to the common combination "vehicle.g".

The model tree can now be walked and "prepped" in parallel, on multi-CPU systems. For large models, tree walking and database prepping can take a significant amount of time; why not use multiple processors when they are available? This feature applies to **mged** viewing commands (like *e*, *ev*, etc.), as well as to callers of the **librt** routine *rt_gettrees()*. Parallel tree walking works, and provides a significant speedup on parallel systems. For the moderately large database "the Mountain Fortress" (12,076 solids in 3,798 regions, in an area 25 km on a side) on an SGI-4D/240, the table gives the measured speedup factor.

CPUs	Speedup	Elapsed Time (sec)
1	1	392
3	2.15	182
4	2.45	160

Notes on parallel tree walking:

1) Only a portion of *rt_gettrees()* runs in parallel, so 100% speedup can never be achieved. There will be a small amount of "serial bottleneck".

2) In the parallel portion, database I/O is involved, resulting in about 20% system time overhead per processor.

The new tree-walker provides better bounding RPP support. When a region contains non-union operations between solids, sometimes better model (and region) bounds are found than with the previous tree walking algorithms. The improved algorithm which is now used is due to Ed Davisson. Most noticeably, this should give much better "view auto-sizing" in **rt**.

The new tree-walker also provides uniform support of articulated animation. Animation directives are now handled automatically by the tree walker, so any program that uses the new tree walker and desires animation capability can now easily obtain it. In particular, this will make it easy for **mged** to preview the articulation scripts that **rt** has been able to render.

1.10. RT Family of Programs

New command line options have been added to all of the **rt** family of programs (e.g. **rt**, **rtcheck**, **rtxray**, **rtbscat**, **rthide**, etc.). First, the "-c" option allows any **rt** animation script command to be given on the command line. Animation script commands which contain embedded spaces need to be enclosed in quotes. For example,

```
rt -s64 -c "set bounces=7" -c "set" moss.g all.g
```

allows the maximum number of ray bounces to be changed (through the first -c option), and the current viewing-model variables to be displayed (using the -c "set" option). Another example is:

```
rt -s64 -c "set background=.25,0,.5" moss.g all.g
```

which sets the background to the old purple background color, using normalized intensities in the range 0 to 1. This can also be accomplished with a shorthand option, using RGB values in the range of zero to 255:

```
rt -s64 -C 63/0/127 moss.g all.g
```

Note that any non-numeric characters can be used as value separators. The comma and slash are the two most often used. The convention generally followed is to use slashes for RGB values, and commas for normalized intensities in the range of 0.0 to 1.0, although the number parser does not care.

The -c option will be especially useful for the radar codes built as **rt** view modules, as it will now allow access to all the application-specific variables from the command line. Traditionally, they could only be set in an animation script, which was not always convenient.

The default background color in **rt** was changed from purple (63/0/127) to black (0/0/1). The choice of 0/0/1 rather than 0/0/0 is so that the checkpoint/restart code can tell the difference between unwritten pixels in the middle of the image and pixels which have been determined to have background color. When **rt** is running in parallel, and when **remrt** is running with more than one server machine, the output file is written in semi-random order. As each span of pixels is completed, it is written to the appropriate location in the output **pix**(5) file. If this results in intermediate portions of the file not being written, UNIX will zero-fill (with 0/0/0) those portions automaticly. By choosing the background color to be 0/0/1, this saves **rt** from having to recompute all the rays that result in background color pixels on a restart.

The choice of 0/0/1 over other dark colors was made because the eye is less sensitive to blue than to either red or green. Keen-eyed watchers would be less likely to see 0/0/1 than, say, 0/1/0. Ordinarily, 0/0/1 (which is 0,0,0.0039 in floating point format) is indistinguishable from black 0/0/0. Compositing images generated with the new background color can be done with:

```
pixmatte foo.pix -C0/0/1 stuff
```

A number of new ray-tracing analysis programs based on the "rt view module interface" [Muuss91b] have been added, including **rtxray** for making X-ray images of models, **rtbscat** for computing radar backscattering, and **rthide** for producing hidden-line-removed UNIX-plots of models [Muuss91c].

1.11. REMRT

The network distributed ray-tracing software was enhanced to suffer less from dispatcher latency, so that significantly more compute servers can be employed [Muuss87a]. Also, the algorithm for determining server assignment size has been made more dynamic, now taking into account both the measured performance of the compute server, and the workload of the dispatcher process. Also, the size of the maximum assignment was significantly increased, to take advantage of today's faster computers [Muuss90b].

remrt's distributed raytracing server, named **rtsrv**, has gained a new feature. When starting up, it reads the file /usr/tmp/public_cpus to find out how many CPUs on that system it may use. The file contains a single number, in ASCII, on a line by itself. If the file does not exist, **rtsrv** will create it. This file is writable to all users, and is is expected to be updated at any time by the users of systems running **rtsrv**, to exert control over the load on their machine.

If the number of public CPUs is zero, **rtsrv** will not continue to run on that machine. If the number of public CPUs is positive, **rtsrv** will use that many CPUs, not to exceed the number of physical CPUs actually operating at that time. If the number of public CPUs is negative, **rtsrv**

will use "all but" that many CPUs. For example, if public_cpus = -2 and the system has 8 CPUs, **rtsrv** will use 6 ("all but 2"). These semantics follow **rt**'s -P (number of processors) option. This makes it possible to use some, but not all, the resources of a given machine for **rtsrv**.

1.12. Cake & Development Environment Support

The "cakeinclude.sh" Shell script was replaced with a new C program "cakeinclude" by Lee Butler, which is compiled and installed automaticly by "setup.sh". This provides a significant speedup when **cake** builds it's dependency lists, each time **cake** is run. Here are some timings for running on a full directory of source. The speedup factors are computed from the decrease in elapsed time. (SWM is a diskless Sun-3/50, SPARK is a Gould PN9080, and VOYAGE is an SGI 4D/240. The files were accessed via NFS from SPARK).

Machine	SrcDir	Script	Program	Speedup
swm	rt	4.7u 7.0s 1:00 19%	2.1u 2.7s 0:11 43%	5.5X
swm	util	12.2u 22.5s 2:48 20%	5.1u 8.8s 0:24 58%	7.0X
spark	rt	2.9u 7.2s 0:17 59%	1.2u 2.5s 0:05 65%	3.4X
spark	util	7.8u 23.4s 1:00 51%	3.5u 9.1s 0:21 58%	2.9X
voyage	rt	1.2u 3.0s 0:05 66%	0.2u 1.0s 0:02 37%	2.5X
voyage	util	3.2u 11.0s 0:20 68%	1.2u 4.0s 0:10 48%	2.0X

The location of the include files has changed. Release 3.7 uses /usr/include/brlcad. Release 4.0 uses /usr/brlcad/include, with a symlink from /usr/include/brlcad. The intention is to keep all files related to the BRL-CAD distribution located in one place in the filesystem tree. References to these files will be inserted into the appropriate system directories (/usr/include, /usr/lib) using symbolic links.

1.13. Release 3 of Utah Raster Toolkit

The latest version of **librle** was installed, as have all the new image handling tools from the Utah Raster Toolkit. All BRL-CAD utility programs have been converted to use the new libraries. Although **librle** usage has been changed slightly by Utah, the **rle**(5) file format has not changed.

1.14. Improvements to Utility Programs

pix-fb, **bw-fb**, and **pixhalve** all gained the "-a" flag, which invokes a file *autosize* function. A table of common image sizes is consulted, and if the input image size matches the size found in the table, then the values for width & height from the table are used. This goes a long way to keeping the "headerless" image file formats convenient to use, given that BRL-CAD is now often used with images other than the default size of 512x512. Other programs that deal with images will probably also acquire this flag.

W	N	Format
50	50	
64	64	
128	128	
160	120	1/4 640x480
256	256	
320	200	PC screen
320	240	1/2 640x480
512	512	(default)
640	512	1/2 1280x1024
640	400	PC screen
640	480	SGI NTSC
720	486	Abekas NTSC
1024	768	SGI-3D screen
1152	900	Sun screen
1024	1024	-h
1280	960	2* 640x480
1280	1024	SGI-4D screen
1440	972	2* Abekas
2048	2048	
4096	4096	
8192	8192	

Image Sizes Supported by Autosize

pixhalve is a program to downsample an image to one-half it's original size. It does so using a 5x5 filter kernel to obtain each pixel in the output file. The intent was that a single-pixel wide image feature in the input image will be guaranteed to influence at least two pixels in the output image, even though the output image has only one quarter the number of pixels in it. This becomes quite significant when producing images for NTSC video, because the NTSC chrominance bandwidth is only half that of the luminance bandwidth. As a consequence, the color can change only *every other* pixel [Conrac80a]. When **pixhalve** takes a 1280x960 image and reduces it to 640x480, displaying the resulting image on a composite NTSC monitor produces a rendition of the image that is about as good as NTSC can get. This defines a new high point in BRL-CAD's ability to prepare images for NTSC video display.

1.15. Halftone package

While the majority of image synthesis done with the BRL-CAD Package results in 24-bit color images, the most common format for communicating results is still the (1-bit deep) monochrome printed page. The halftone package provides a sophisticated set of image contouring, sharpening, and halftoning routines to produce high-quality one-bit monochrome images which retain most of the visual information of the 24-bit original. Because information is being discarded, the user will have to experiment with the various options to determine which processing modes reproduce the image with the best fidelity [Johnson91a].

1.16. Minimal Standard Random Number Library

An implementation of the minimal standard random number generator described by Park and Miller in CACM [Park88a] as been included as **libmsr**. Both integer and floating point uniform random numbers are provided, as well as floating point Gaussian random numbers. This library computes a large array of random values and then provides them as required via a fast macro. The array is refilled when all the values have been used. This permits the creation of the random numbers to be vectorized, and keeps the subroutine linkage overhead to a minimum.

1.17. EDPIX

An SGI-specific mouse driven picture editor and "paint" program called **edpix** was added, to supplement the **fbed** program, which has fewer features but is vendor independent.

1.18. NIRT

A new program was added to the distribution for interactively exploring a model by firing single rays at the model from keyboard commands. The list of region partitions that the ray passes through is printed on **stdout** [Tanenbau91a]. This can be a valuable tool for checking models, and for debugging ray-tracing applications codes.

2. Future Directions beyond Release 4.0

2.1. VINT GUIMS

As a result of carefully designed file formats, all BRL-CAD image files and plot files are network transparent and vendor independent [Muuss90b]. This allows the full power of network computing to be easily applied to most image processing tasks. Because of the clean and elegant UNIX "filter" and "pipeline" strategy employed, existing tools can be combined in many configurations, allowing most problems to be solved without writing additional programs [Muuss88a]. Yet, at the same time, creating a new image processing tool in C takes only a few man-days to program. This is a consequence of the UNIX "Standard I/O" paradigm and BRL-CAD's powerful collection of subroutine libraries.

It should be possible to achieve these same kinds of efficiencies, both in programming and using such tools, while extending the tools to have sophisticated "user friendly" graphical user interfaces (GUIs). Adding a graphical user interface to an application code requires weeks or months of programming. The details of the visual appearance need to be specified in minute detail, and provisions need to be made for handling the wide variety of events which might occur: keyboard events, buttons, knobs, mouse events, window system events, etc.

In 1979, when **ged** (the precursor to **mged**) was first created, the Apple Macintosh computer did not even exist, and the now-famous Xerox bitmapped user interface was just a laboratory curiosity. High-performance graphics display hardware proliferated, and **ged** evolved into **mged**, gaining a clean interface to a vendor-independent *display manager* library to access the display hardware. Shortly after this, Chuck Kennedy used **libpkg** and *htond()* to create an experimental *network transparent* remote display-manager interface. This early experiment was BRL's first encounter with the power and possibilities of a Vendor-Independent Network-Transparent (VINT) Graphical User Interface (GUI). At the same time, **libfb** gained it's network transparent "remote framebuffer" capability. Because of the object-oriented design, any combination of local and remote **libfb** framebuffers could be *stacked* together, and thus the capability for *tele-science* was born. Over the years, this has proven immensely useful, with applications ranging from sending images to remote co-workers, to creating multiple-location demonstrations, to facilitating trans-continental scientific collaboration.

Looking ahead, it should be possible to enjoy all the benefits of the current environment (rapid program development, network transparence, vendor independence, and high performance) in the context of a full-feature graphical user interface. In addition, there should be some tools for context-sensitive help semi-automatically provided, which expands the GUI into a graphical user interface management system (GUIMS). For example, it should be possible to take an existing image processing tool, and with no more than an additional few days of programming effort, add a sophisticated graphical user interface with online help and a significant number of control parameters and interaction forms. Ideally, this GUI version will still achieve a full color graphical display with interactive performance comparable to that currently achieved with **libfb** or **mged** running on the same hardware platform. This is the goal of the BRL-CAD VINT GUIMS project.

The X Window System [Scheifle86a] is emerging as the "lowest common denominator" vendor independent network transparent window system standard. The X Window paradigm of

preparing bitmaps on the host computer and sending them across the network to the display computer, works superbly for monochrome displays containing text and simple graphics. Editing complex 3-D geometry as is done with **mged** requires a tremendous bandwidth between **mged** and the display screen. For local display on a workstation, it seems likely that using vendor-specific software to drive the graphics display hardware will give better performance than using X. For example, in SGI's IRIX Release 4, user interface operations will be handled in X, but high performance graphics applications will still have to be written using SGI's proprietary **libgl**. Also, porting X to a new workstation requires substantial effort; it is likely that supporting the VINT GUIMS directly on a new workstation would be much simpler than porting X.

A successful VINT GUIMS implementation will result in an immediate improvement in the user interfaces of **mged**, MUVES, and the BRL-CAD image processing tool suite. A prototype VINT GUIMS effort is currently in progress as a joint collaboration between BRL and The Royal Melbourne Institute of Technology (RMIT).

2.2. New MGED UIF, using VINT GUIMS

With the addition of NMG support to **mged**, a rich set of possibilities for improved geometry editing become apparent. For example, many existing faceted primitives can be converted to NMG form without loss of information. Thus, if sophisticated new interactive editing facilities which operated exclusively on NMGs were implemented, this investment could also produce enhanced editing capability for many existing solids with minimal additional effort. By employing the concept of "editing under constraints", this facility could be easily extended to editing NURBS as well.

Adding a significantly more complex editing capability to **mged** will require a tremendous amount of additional user interaction. In the context of the current **mged**, much of this interaction would be via the traditional keyboard-oriented command set and could prove cumbersome to use. However, if even part of the promise of the VINT GUIMS is realized, then a comprehensive new user interface for **mged** could be created, entirely in the context of the VINT GUIMS.

2.3. Method of Moments

When a metallic vehicle is illuminated with radar energy, that energy is partly absorbed, and partly dispersed back into the surroundings. Some of the illumination energy returns to the transmission position, and it does so carrying an electronic "signature" of the vehicle [Toomay82a]. A variety of different techniques exist to calculate the predicted radar signature of a given vehicle. The algorithms based on ray-tracing tend to handle multi-bounce effects very well but are unable to simulate edge diffraction and creeping wave phenomena. However, algorithms based on feature-based descriptions of the the vehicle or coarse polygonalizations tend to handle diffraction and creeping waves acceptably but are unable to handle multiple bounce effects. The best known technique for the simulation of radar signatures is the Method-of-Moments technique [Harringt82a, Pizer84a] , which requires a polygonalization of the surface of the vehicle as input.

In order to achieve high accuracy, the Method-of-Moments technique requires that each surface polygon be no wider than one fifth of one wavelength of the radar signal. The relationship between frequency f and wavelength λ is given by

$$\lambda = \frac{c}{f} = \frac{3 \times 10^8 \ \ m/s}{f \ \ Hz}$$

Thus, the method of moments technique requires exceptionally fine surface tessellations to be used. Tessellating full size vehicles this finely produces a gargantuan number of facets. This is one of the prime applications of the NMG tessellation capability. An experimental Method-of-Moments processing pipeline was implemented and is undergoing testing. An experimental **mged** command is used to capture the tessellation of a model as a binary polygon file. These polygons are converted to records containing centers, normals, and areas (CNA file). The CNA file is processed to build a complex Z matrix. The Z matrix is inverted, to produce the complex invZ matrix. The incident field is computed, and the field is multiplied by the invZ matrix to produce

the solution field. The solution field is applied to the polygon file to produce a colored polygon file, which is then rendered.

2.4. Interface to MESA, CTH

Both MESA [Mandell89a] and CTH need only volume-fraction per cell information. Computing this volume fraction information is a good first step towards the goal of producing full three-dimensional Finite-Element-Mesh (FEM) data directly from the geometry database. This will create an important linkage between BRL-CAD's ability to model complex shapes, and a very important family of structural analysis codes.

Extracting volume fractions from an **mged** database is very straightforward, and the first implementation will be done using the ray-tracing paradigm. For comparisons of speed and accuracy, an NMG-based implementation will also be attempted.

2.5. librt performance from NUGrid

At AUSGRAPH '90, Mike Gigante presented a new space partitioning algorithm dubbed NUGrid [Gigante90a], which is currently the best known method for efficiently handling the "teapot in a stadium" problem. An experimental implementation of the NUGrid algorithm exists for **librt**, and the hope is that when completed, this will provide a significant performance improvement over the non-uniform binary-tree algorithm that is currently in used in **librt**.

In addition, the NUGrid algorithm has also been successfully employed in an application which solves the 3-dimensional cluster analysis to determine the minimum number of perceptually "close" pixel values used in a full-color image. This is an important step in converting images for display on color framebuffers that are only 8 bits deep, such as is commonly found on personal computers and low-end workstations like Suns [Johnson91a].

2.6. Trimmed NURBS support

Trimmed NURBS (t-NURBS) are a relatively recent development, circa 1986. Finding the intersection of two NURBS surfaces in 3-space is very difficult. However, once the intersection curves have been found, if the solution is recorded as a pair of curves in the u,v parameter spaces, then using the intersection curves becomes very easy. Once the intersections are found, the trimmed NURB is used to cut out the unwanted regions, and then the two surfaces are jointed together.

As an extra benefit, capping operations become trivial, even if the extruded shape to be capped has a very convoluted contour. In this case, a rectangular NURB is placed on top of the extrusion, and the projection of the end surface onto the NURB is used as the trimming curve. Now the trimmed NURB forms the perfect cap.

The algorithms for implementing t-NURBS have been designed and implemented, and testing is underway.

2.7. Combining NMGs and Trimmed NURBS

An NMG representation comprised exclusively of rectangular parametric surfaces, such as B-splines or similar tensor-product surface patches, was considered. However, research to date has shown that while B-spline surfaces can be combined using boolean operations, the resulting object can not be expressed strictly in terms of B-splines [Thomas84a]. Existing algorithms for intersecting two NURBS surfaces have returned the results as a collection of many smaller NURBS, plus bundles of polygons from the vicinity of the intersection curve. This occurs because the boolean combination of rectangular parametric surfaces is not necessarily bounded by rectangular parametric surfaces; i.e., the representation is not closed. This mixed representation of B-splines and polygons becomes ungainly when subjected to repeated boolean operations.

Thus, in the design of the existing NMG system, NURBS can not be used for storing the faces of solids, because the B-spline representation does not have closure under the set of boolean operations. Recent work has suggested that a representation comprised of trimmed B-splines and

shared-edge polylines might be closed under boolean operations [Cobb84a], but a full implementation is not yet known to exist. Therefore, when NMGs were implemented for BRL-CAD, there was no choice: the explicit representation of modeled objects had to be expressed in terms of collections of polygons.

The major significance of the trimmed NURBS work is that trimmed NURBS (t-NURBS) offer the potential of being closed under Boolean operations. If this is indeed the case, then this breakthrough permits full CSG modeling to be performed on trimmed NURBS, without needing any auxiliary data structures or surface representations.

The implication of this is tremendous. It means that the existing NMG implementation, which only supports solids with planar faces, can be readily extended to include two kinds of faces: planar faces, and t-NURBS faces. In such a t-NURBS extended NMG implementation, the power of free-form surface modeling can be employed on those surfaces which require it, while planar faces can retain their original simplicity.

This provides a natural path for adding detail to geometric models later in the design cycle. Imagine a vehicle designer starting with a front glacis plate modeled as an ARB8: a slab with 8 vertices and 6 planar faces. As the design evolves, imagine that the outer face of the slab needs to be thickened somewhat in the center. In a system which employed a homogeneous spline representation, all six faces would have to be converted to splines. With t-NURBS extended NMGs, only the outer face needs to be upgraded into a t-NURB. The four edges on that face become space curves, while the remaining five faces are kept as simple planar faces, albeit with the four side faces also being modified to have one curved edge and three linear edges each. This permits tremendous economies.

2.7.1. Storage Efficiencies

NURBS take significantly more storage to represent than planar faces. While superficial consideration reveals that a rectangle in the plane can be represented efficiently with NURBS, this efficiency does not persist when the planar NURBS faces need to be interfaced to genuinely curved faces. In the example given above, the non- t-NURBS implementation has to convert each side face into a spline with one simple knot vector, and one knot vector that was compatible with the curved face.

2.7.2. Performance Efficiencies

The time to compute intersection curves between NURBS surfaces increases as the complexity of the surfaces increases. Even though the individual knots used in the NURBS surface are designed to provide only local support of the surface, some increases in processing time are unavoidable, because of the increased amount of data to sift through. If each of the m edges has k_m knots in it, the traditional NURBS approach has complexity roughly proportional to

$$\prod_{i=0}^{i=\frac{m}{2}} k_i$$

while the t-NURBS extended NMG approach has complexity proportional to only

$$\sum_{i=0}^{i=m} k_i$$

2.7.3. Designer Efficiencies

In a full scale solid modeling system, the designer expresses the initial structures directly into the modeling system's editor, just as a modern author creates his "rough draft" directly into a word processor. Most elements of a design begin as "rough", blocky shapes, which are refined into the final form by applying modifications. This strategy is akin to how a sculptor crafts a sculpture: large chunks are removed from the slab of stone until an approximation of the desired shape

is achieved, after which progressively smaller modifications are made until the result is acceptable. Similarly, the t-NURBS extended NMG approach permits the designer to build using simple primitive shapes, confident that any portions of the design which need to be modified can be simply and economicly refined in the future.

2.7.4. Improved tessellation

Another significant implication of the t-NURBS extended NMG approach occurs in the area of tessellation. Many of the existing CSG primitive solids can be tessellated into *exact* t-NURBS representations. This means that computing evaluated surface descriptions of the underlying CSG database can, in many cases, be an exact (albeit voluminous) description of the solid. This allows deferring the conversion of the evaluated surface descriptions into polygons until a later processing stage.

2.7.5. Analytic Surface Calculations

This affords the opportunity to create new analysis codes that can process curved surfaces, yet at least initially only have to deal with one very general kind of shape. This also provides a very direct and natural interface to spline-based [Rogers90a] and Bezier-patch [Bezier74a] based modeling systems.

Given a homogeneous geometric representation such as the Trimmed B-Splines just discussed which also has an analytic representation, a further processing capability arises. Rather than interrogating the database by sampling or subdivision techniques, the direct mathematical manipulation of the source geometry through its parametric representation becomes possible. Calculations of physical properties requiring integration over a surfaces can often be evaluated with greater accuracy using an explicit analytic calculation rather than by numerical evaluation. While this may be difficult in general due to the complexity of the parametric expression, some classes of surface representations are good candidates. Splines, for example, are piecewise-polynomial functions which have relatively simple Fourier transform representations. Since 2-D spatial Fourier transforms arise frequently in far-field electromagnetic scattering calculations, exploitation of the parametric spline representation is of interest in predictive scattering calculations. Direct use of spline parameters in a Physical Optics scattering model is part of the methodology used at the Aircraft Division, Northrop Corporation.

With the rapidly developing potential of symbolic calculation, treatment of seemingly impossible formulas resulting from the geometry/physics interaction may become tenable. This can help to reduce the trend towards employing numerical methods at the onset of a problem and avoid some of the accompanying instabilities and errors.

3. Summary

This is an exciting time in the development of the BRL-CAD Package. On the one hand, the geometric coverage of the modeling system is expanding rapidly, while on the other hand, the interface between the geometry and analysis codes continues to grow richer. This leads to support for an ever-increasing assortment of engineering analysis tools to provide a hitherto undreamed-of level of computer assistance to the designer. One can only speculate about what new accomplishments this will facilitate.

4. Acknowledgments

The BRL-CAD developers would like to gratefully acknowledge the financial support for the NMG work which was provided by the Joint Technical Coordinating Group-Munitions Effectiveness (JTCG-ME), Smart Munitions Working Group, under grants monitored by Julian A. Chernick, US Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, 21005-5066.

The author would like to thank Dr. Paul Deitz for providing unflagging support and encouragement for the BRL-CAD effort. Also, the clarity of the paper was greatly improved thanks to numerous suggestions by Susanne Muuss and Christopher Johnson.

References

Applin88a. K. A. Applin, M. J. Muuss, R. J. Reschly, M. Gigante, and I. Overend, *Users Manual for BRL-CAD Graphics Editor MGED*, BRL Internal Publication (October 1988).

Bezier74a. P. E. Bezier, *Mathematical and Practical Possibilities of UNISURF*, Academic Press, New York (1974).

Cobb84a. E. S. Cobb, *Design of Sculptured Surfaces using the B-spline Representation*, PhD dissertation, University of Utah (June 1984).

Conrac80a. Conrac, *Raster Graphics Handbook*, Van Nostrand Reinhold, New York (1980).

Gigante90a. M. Gigante, "Accelerated Ray Tracing using Non-Uniform Grids," *Proceedings of Ausgraph '90*, pp. 157-163 (September 1990).

Harringt82a. R. F. Harrington, *Field Computation by Moment Methods*, Krieger, Malabar, Florida (1982).

Johnson91a. C. Johnson, *Digital Halftoning: The halftone package and Applications to BRLCAD*, BRL-CAD Symposium '91, Aberdeen Proving Ground, MD (7-9 May, 1991).

Mandell89a. D. Mandell and T. Adams, *MESA Input Manual*, Los Alamos National Laboratory Report LA-CP-89-345 (August 1989).

Molnar87a. Zsuzsanna Molnar, "Advanced Engineering/Scientific Graphic Workstations," in *Techniques for Computer Graphics*, ed. D. F. Rogers, R. A. Earnshaw, Springer-Verlag (1987).

Muuss89a. M. Muuss, *Frame Buffers and Window Systems: Internals of the Silicon Graphics 4D Framebuffer Support for LIBFB*, BRL-CAD Symposium '89, Aberdeen Proving Ground, MD (24-25 October, 1989).

Muuss87a. M. J. Muuss, "RT and REMRT - Shared Memory Parallel and Network Distributed Ray-Tracing Programs," *USENIX: Proceeding of the Fourth Computer Graphics Workshop*, (Oct 1987).

Muuss88a. M. J. Muuss, "Distributed Graphics for High-Resolution 3-D Modeling," *SGI Iris Universe Magazine*, pp. 14-19 (Fall 1988).

Muuss90b. M. J. Muuss, "Workstations, Networking, Distributed Graphics, and Parallel Processing," in *Computer Graphics Techniques: Theory and Practice*, ed. D. F. Rogers, R. A. Earnshaw, Springer-Verlag (1990).

Muuss90a. M. J. Muuss and L. A. Butler, "Boolean Operations on Boundary Representation Solids Using n-Manifold Geometry," *Proceedings of Ausgraph '90*, pp. 291-299 (September 1990).

Muuss91a. M. J. Muuss and L. A. Butler, "Combinatorial Solid Geometry, Boundary Representations, and Non-Manifold Geometry," pp. 185-223 in *Advanced Computer Graphics Techniques*, ed. D. F. Rogers, R. A. Earnshaw, Springer-Verlag (1991).

Muuss91b. S. Muuss, *Rapid Creation of Analysis Codes: The RT View-Module Interface*, BRL-CAD Symposium '91, Aberdeen Proving Ground, MD (7-9 May, 1991).

Muuss91c. S. Muuss, *RTHIDE: 3-D Hidden Line Removal*, BRL-CAD Symposium '91, Aberdeen Proving Ground, MD (7-9 May, 1991).

Park88a. S. Park and K. Miller, "Random number generators: good ones are hard to find," *Comm. ACM* **31**(10)(Oct 1988).

Pizer84a. J. Moore and R. Pizer (eds), *Moment Methods in Electromagnetics*, Wiley, New York (1984).

Rogers90a. D. F. Rogers and J. A. Adams, *Mathematical Elements for Computer Graphics, 2nd ed.*, McGraw-Hill, New York (1990).

Scheifle86a. Robert W. Scheifler and Jim Gettys, "The X Window System," *Transactions on Graphics* **5**(2) pp. 79-109 (April 1986).

Shoemake85a. K. Shoemake, "Animating Rotation with Quaternion Curves," *Computer Graphics* **19**(3) pp. 245-254 (July 1985).

Shoemake89a. K. Shoemake, "Quaternion Calculus for Animation," pp. 187-205 in *Course Notes: Math for SIGGRAPH*, ACM SIGGRAPH (1989).

Tanenbau91a. P. Tanenbaum and N. Eberius, *NIRT: Interactive Ray-Tracing with BRL-CAD*, BRL-CAD Symposium '91, Aberdeen Proving Ground, MD (7-9 May, 1991).

Thomas84a. S. W. Thomas, *Modelling Volumes Bounded by B-spline Surfaces*, PhD dissertation, University of Utah (June 1984).

Toomay82a. J. C. Toomay, *Radar Principles for the Non-Specialist*, Lifetime Learning Publications, London (1982).

The BRL-CAD Package
An Overview

Phillip C. Dykstra
Michael John Muuss

Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

ABSTRACT

The major components of the BRL-CAD Package are reviewed. The BRL-CAD Package is a combinatorial solid geometry (CSG) based modeling system which includes an interactive model editor, a ray tracing library, a generic frame buffer library, and a large collection of related tools.

An object-oriented ray tracing library provides the primary method of model interrogation. A whole family of engineering analysis applications based on the ray tracing paradigm has been built, including traditional renderers, and predictive radar models. A generic frame buffer library interface with transparent networking capability provides hardware independent access to any display device from any host. Several categories of software tools for image display, manipulation, and analysis are discussed. Some general user interface issues are mentioned.

This paper emphasizes the reasons which led to the system as it exists today, and comments on some of its various strengths and weaknesses.

The BRL-CAD Package
An Overview

Phillip C. Dykstra
Michael John Muuss

Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

1. Introduction

The Ballistic Research Laboratory CAD Package is a large body of software consisting mainly of 1) a solid model editor (MGED), 2) a ray tracing library for model interrogation (librt), 3) a generic frame buffer library with full network display capability (libfb), and 4) a large collection of software tools for frame buffer and image manipulation and analysis. Parts of this system have roots in work done over two decades ago, most notably the solid modeling, and the ray tracing. Recently this software has been through a new generation of growth. It is now distributed free of charge to many sites around the world on a non-redistribution basis.

As with many large systems, parts of it were the result of years of evolution, with many band-aids, hacks, and "backward compatibility" requirements along the way. The work that one needed to accomplish today was often more influential than any carefully made plans. Most of this history is known only to those who watched it happen.

This paper provides a brief overview of the major components of the BRL-CAD system. It will attempt to explain how and why many parts of it are the way they are. Finally, it will entertain the question of what is good and bad about it, and how the various decisions that were made have or have not worked.

2. Solid Modeling - MGED

The BRL has been building solid models of vehicles and other objects for over twenty years. These models are analyzed for various physical properties (such as center of mass, moments of inertia), vulnerability, and more recently for optical, radar, and IR signatures.

This work began in the early 1960s when BRL had the Mathematical Applications Group Inc. (MAGI) develop a method of geometric description for military vehicles.[1] The method decided upon was Combinatorial Solid Geometry (CSG), sometimes referred to as Set Theoretic modeling. This is a system where various geometric solids (boxes, cones, ellipsoids, tori, etc.) are combined using boolean operations (union, intersection, and subtraction). CSG represents one of the two major classes of modeling, the other being surface or boundary representations (B-reps). A key reason for the selection of CSG modeling is that it is "true to reality." Physical objects are solids, not just surfaces. If an object has been constructed with CSG, one is at least assured of its physical possibility.

For several years, models were constructed on large sets of punch cards. One or more cards would contain the parameters for a particular solid; other cards would describe the boolean relationships between solids. This system was not hierarchical, all solids and combinations existed at one level. Ray tracing was used to analyze these models, but the only images of these models ever produced were crude plotter drawn wire frames.

A new generation of modeling tools emerged in 1979-1980. A system was built which allowed these models to be interactively displayed and edited on vector display devices. The

success of these early efforts, coupled with the failure to find commercial tools of sufficient power, led to the development of the MGED model editor. The MGED editor is written in C and has been run on a large variety of machines. An object oriented interface to a set of display managers allows many different display devices to be supported. The types of primatives supported include: arbitrary boxes of up to eight vertices, ellipsoids, truncated general cones, tori, polygonal solids, and solids constructed of B-spline surfaces.[2]

The CSG representation is a natural form for our most common method of model interrogation - ray tracing. There are some methods of analysis however for which a surface facet representation of a model is the desired form. Work is currently under way on the facetization of CSG models, in order to support the needs of such codes. Future work is also planned in automatic mesh generation for similar reasons. These two capabilities will further ease the barrier between model representation, and model analysis.

For a much more comprehensive coverage of solid modeling, with MGED as a case study, see Muuss.[3]

3. Model Analysis - Ray Tracing

Ray tracing is a method of point sampling a geometric model by mathematically intersecting lines with objects in the model. At each intersection point various properties of the model can be determined: where did it intersect, what is the surface normal and curvature at that point, what part of the model was hit, what are the material properties at that point, etc. The computer graphics community often cites the origins of ray tracing with Kay's 1979 thesis,[4] or Whitted's paper of 1980.[5] However, the use of ray tracing as a method of geometric model interrogation has its origin in a BRL contract with MAGI, the initial results of which were published in 1967.[1] More details on the origins of ray tracing can be found in Muuss.[6] For an overview of the method itself, see Rogers.[7]

Ray tracing is the primary method used by BRL for model interrogation. While ray tracing (and radiosity) is generally held as the highest fidelity form of rendering (image generation) available, many people in the computer graphics community dislike it due to its notoriously high computational expense compared to other rendering techniques. But there are several key reasons why BRL uses it: 1) We are primarily concerned with doing an engineering analysis of the model, not just making pretty pictures of it, this objective is what led us to CSG models to begin with. 2) When CSG models are used, ray tracing is the most common method for evaluating the boolean expressions, 3) Firing a ray at a model is very much like firing a projectile (or light) at it, and is thus a natural method for vulnerability and signature analysis.

The ability to intersect rays with a model is common to all of the analysis tools, whether one is rendering a picture of the model or computing a moment of inertia. For this reason, the code which knows how to efficiently trace rays through a CSG model has been put in a library, librt. An application linked to this library has complete control over which rays are fired, how much information is computed at the intersection points, and what is done with the returned information. This library level separation of ray tracing and analysis has proven to be an extremely good one.

Other splits between ray tracing and analysis have been made or proposed. Some systems trace the entire model, placing the results into an intermediate file. There are two problems with this: the analysis code can not influence the ray trace (for example, by deciding when to reflect or when to fire extra rays in an area), and the volume of data generated is extremely large, often filling an entire large disk drive. The split could also be implemented by passing messages between separate processes via a remote procedure call, or a stream mechanism such as a UNIX pipe. The amount of overhead involved with either of these methods is typically of the same order of magnitude as the work involved in tracing a single ray. This approach is thus felt to be impractical.

Two ray tracing programs which use librt are provided in the CAD package: RT and LGT. LGT is an optical rendering program with a *curses* based screen oriented user interface. RT also provides rendered images with command line arguments, but is itself the front end for several

applications including a radar model. RT also has the ability to read scripts of commands which can control the computation of a sequence of frames, and the orientations and properties of materials in each frame of an animation.

Future work with the ray tracer includes extending the classes of traceable objects, further efficiency improvements, and its extension to handle a broader class of physical phenomena. The latter goal includes multiple spectral point sampling (instead of just Red Green Blue) to account for dispersion and complex spectra, divergence factors (for the concentration and diffusion of light), and polarization effects.

4. The Frame Buffer Library

The frame buffer library (libfb) provides a device independent interface to a raster display. A program compiled with this library can access many different display types, including those on other machines on the network. The most important routines are summarized below.

libfb routines	
fb_open(device,width,height)	open the device
fb_close(fbp)	close the device
fb_read(fbp,x,y,buf,count)	read count pixels at x,y
fb_write(fbp,x,y,buf,count)	write count pixels at x,y
fb_clear(fbp,color)	clear to an optional color
fb_rmap(fbp,colormap)	read a color map
fb_wmap(fbp,colormap)	write a color map
fb_window(fbp,x,y)	place x,y at screen center
fb_zoom(fbp,xzoom,yzoom)	pixel replicate zoom
fb_getwidth(fbp)	actual device width in pixels
fb_getheight(fbp)	actual device height
fb_cursor(fbp,mode,x,y)	cursor in image coords
fb_scursor(fbp,mode,x,y)	cursor in screen coords
fb_log(format,arg,...)	user replaceable error logger

The coordinate system for x,y specifications is first quadrant. While we went round and round about first vs. fourth quadrant with arguments akin to "which end of the egg first," the decision for first quadrant resulted primarily because that is the same ordering as our image files (.pix files, see below). The image files themselves were ordered that way because the University of Utah's RLE files are first quadrant. If reads and writes extend beyond the end of a scanline, they wrap in first quadrant fashion.

The pixels passed to and from the library are simply arrays of bytes interpreted as RGBRGB.... While we used to define a C language pixel structure with red, green, and blue elements, this was changed to a typedef'd array of three unsigned chars. This was important in order to avoid structure padding. The Cray computers for example would have used eight bytes per pixel with the old format. Unfortunately, one does run into some compiler touchiness when using pointers to typedefs which are themselves arrays!

The display to be used is selected by a command line argument, an environment variable FB_FILE, or a default for the system the code is running on. The format is [host:]/dev/*device_name*[#], or simply "filename". The /dev/ part is used to identify a display device. The *device_name* need not correspond to entries in /dev, it is just that if the /dev prefix is not given a file pathname is assumed. If a hostname is given, a network connection is opened to the frame buffer library daemon (rfbd) on that machine. The remaining part of the string is passed to that host for the open (this generalizes the open to allow multiple "hops" in order to get to a host). Currently supported displays include the Adage Ikonas, Silicon Graphics Iris, black and white and color Sun workstations, Raster Technologies One/80, the X Window System, and AT&T 5620 terminals. There is also a debug interface, a disk file interface, and a "stack" interface that allows multiple output devices to be combined together.

A set of buffered I/O routines is also provided. In this interface a "band" of scanlines is kept in memory and the appropriate pre-reads and flushing is done. While this interface can speed up single pixel reads and writes, it does not make the drawing of vertical lines any easier, since such a line would run through several bands. In practice, very few of our programs use buffered I/O. Most programs keep their own scanline buffers and do unbuffered scanline size reads and writes. Some thought has been given toward allowing the selection of the memory buffering mode at run time, perhaps keyed on a device name parameter. This would permit the user to control the trade off between speed and interactive output. The ability to make such a decision becomes particularly important when one is using a remote display.

libfb buffered I/O	
fb_ioinit(fbp)	set up a memory buffer
fb_seek(fbp,x,y)	move to an x,y location
fb_tell(fbp,xp,yp)	gives the current location
fb_rpixel(fbp,pixelp)	read a pixel and bump location
fb_wpixel(fbp,pixelp)	write and bump current location
fb_flush(fbp)	bring display up to date

The frame buffer library owes much of its current form to its history. One of the first true frame buffers purchased by BRL was an Ikonas (now Adage RDS-3000), in 1981. This device runs as either a 512x512 or 1024x1024 display with 24 bit pixels. It has three 256 entry 30-bit (10 bits per DAC) color maps, hardware pan and zoom, and hardware cursor support. Michael Muuss of BRL wrote our first library for that device (libik).

Later, a Raster Technologies One/180 frame buffer was acquired and a libik like interface was created for it. As other devices followed, libfb was born. At first there was a switch in every library routine for every display device. Later it was reworked to have an object oriented interface: opening a device fills in a function switch table with that display's routines, and a "frame buffer pointer" was returned to that structure. Most of the frame buffer routines became macros which vector directly out to the device dependent code.

Finally, the machine which had our nice displays on it (a VAX 11/780) was also one of our slowest. To make this less of an issue, a libfb look alike was put together one evening which passed all library calls and returns across a network connection to a daemon that made calls to a "real" libfb. This was facilitated by the Package Protocol[8] (PKG) which allows messages to be exchanged, both synchronously and asynchronously, across a TCP connection (this protocol had originally been developed to make a remote MGED display possible, but later found uses in command and control experiments, etc.). The remote frame buffer code was merged into libfb during its object-oriented restructuring, so that one need only link with a single library to get both local and remote display capability.

Starting with the Ikonas in some sense spoiled us. It gave us full color pixels, color maps, cursors, and pan and zoom. These features were incorporated into the generic frame buffer model used in our library. This makes fitting devices like the Sun workstations into our library quite trying, but this difficulty is more the result of things that workstations like the Sun can't do than it is a design problem with our library. On the other hand, the Ikonas also left us with programs that have to open the device in one of two "modes", either high or low resolution. To make matters worse, it does not allow the current display mode to be read back from the hardware. Therefore, the open must set the Ikonas to a known state. As a result, every frame buffer program, even those which have little to do with display size (such as those which read or write color maps), carries around a "hires" flag so the device can be opened in the proper "mode."

One commonly asked question is whether a network window system, such as the X Window System[9] from MIT will make the BRL frame buffer library unnecessary. I believe that the answer to this is no. A window system provides a user interface to a workstation (and programs). A frame buffer library provides a programmatic interface to a particular kind of hardware device. A window system can be viewed as a shared resource manager, the resources being the screen,

keyboard, mouse, etc. A frame buffer is usually a dedicated device typically with hardware pan, zoom, and color maps, and usually with little or no general processing power of its own. Frame buffers make poor platforms for a window system interface like X - they would need a server running on the host processor of the system they are attached to, and they tend to be very slow at the "BITBLT" operations that window systems use heavily. Window systems in turn, provide poor interfaces to frame buffers.

We have implemented libfb interfaces for several window systems: MEX and 4sight on Silicon Graphics workstations, SunView on Sun Microsystems workstations, and the X Window System (which will work on any machine having an X server). While window systems such as these seem to be what the industry has to offer to graphics users, all of them are of questionable value as a replacement for a frame buffer. Only a few window system platforms offer 24-bit color, fewer still provide genuine color maps for their windows (where the color map is not consumed mapping say, 8-bit to 24-bit color), and none that the author is aware of support built in pan and zoom. As an example of a more deep seated issue, consider that in a frame buffer when one writes data into it, it stays there until different data is written. In a window system, a window is opened by a program and written in, and typically goes away along with the data after that program has exited. You can't get the data back out of the no longer existent window. We have used shared memory in the host machine where possible, to mimic the permanent store of a frame buffer, as well as creating "lingering" windows which live beyond program execution. Even more difficult is the case of X, where the programs on different machines can rendezvous at a frame buffer "window," but where now should the permanent image data be stored (the server can't do that for you)? Thus the frame buffer library can and is being used under window systems, but window systems were never intended to behave the same way as frame buffers (nor probably should they).

5. The Software Tools

A large number of simple tools for manipulating images and frame buffers are provided in the CAD package. They have been written in the traditional UNIX Software Tools fashion: each performs a simple basic function, with a minimum of back talk, and is intended to be hooked together with other tools to achieve an overall goal. A fair amount of effort has gone into making a standard interface to the tools. All tools provide a usage message if executed with no arguments (often after checking for a tty on stdin or stdout when it expects binary data), and common collection of flags is defined for all of the tools.

The use of software tools for computer graphics is not new. Recent systems advocating this tools based approach include those of Duff[10] and Peterson.[11] The BRL-CAD Package has proven to be extremely flexible as as result of this approach. Generally, a new tool is added whenever the existing ones are found to be inadequate. Success can be claimed if one can easily achieve day-to-day tasks without having to write specialized programs.

5.1. File and Image Formats

Several kinds of files are read and generated by programs in the CAD package. These include model databases in a binary form (with a typical filename extension of .g), portable ASCII versions of those (.asc), and University of Utah Run Length Encoded (RLE) images (.rle). By far the most common image format for the tools however is either eight bit per pixel black and white (.bw) or 24-bit per pixel color (.pix). The files have the simplest format imaginable: there is no header at all, and pixels run in first quadrant order - lower left corner, across the scan lines, bottom scan line first, up through the top scan line. The values in the bytes are viewed an intensities from 0 (off), through 255 (full on). The color (.pix) files are in RGBRGB... order. Note that while we use the University of Utah RLE format, we view it simply as a means of image compression, unlike Utah which actually manipulates RLE files directly in their Raster Toolkit.[11]

The use of a simplistic headerless image format is perhaps the most debatable decision we made. It's primary advantage comes when piping several tools together. Each program is simply handed data. It doesn't have to know "how" to read it; there is no header to discard, or harder still, it doesn't have to do the "right thing" with the header information. Doing the "right thing"

is extremely complicated if the header contains very much information. We have also avoided the N^2 problem of format conversion by converting all other formats into and out of this simple one.

Having "raw" headerless data has its price however. It is difficult to tell whether a given image is color or not, what its dimensions are, etc. File naming conventions (.bw or .pix) solve the first; "standard sizes" of 512x512 or 1024x1024 (hires) help alleviate the second (recall that these came from the Ikonas frame buffer). Note that usually only the scanline length needs to be known, the number of lines can then be found by the file size. Many algorithms simply run until all of the data is gone, and some don't even care about scanlines at all.

5.2. Format Conversion

Several other image formats are accommodated by "filters" that convert one into the other. A selection of these is listed in the table. In all of the tables given the reverse conversion is omitted, e.g. there is also a pix-rle for converting color images into RLE format. Also, only the color (pix) version of a tool has been shown while most have black and white (bw) equivalents. Most of the tools listed also allow a wide variety of options. The color to black and white converter for example (pix-bw), allows either equal, NTSC, or "typical" CRT weighting to be applied. It also allows arbitrary weights to be given for selecting or mixing of the color planes in any way desired.

Selected Format Conversion Tools	
g2asc	model database to portable ascii form
bw-pix	black and white to color image
bw3-pix	three black and whites to color RGB
rle-pix	Utah's RLE format to color image
ap-pix	Applicon Ink-Jet to color image
sun-pix	Sun bitmap to color or black and white
mac-pix	MacIntosh MacPaint bitmaps to color

5.3. Frame Buffer Tools

We have chosen to do most of the image manipulation and processing either on data streams, or on disk files. This was done in order to separate the notion of a device from image handling. A common beginning or end of a processing pipeline is to get or put an image into or from a frame buffer. Frame buffers do allow one to manipulate images in many useful ways however, so some device independent tools are provided for that. These include tools to allow changing color maps, panning and zooming through an image, labeling, etc. Where tools require the user to move a cursor or the image, both EMACS and VI style commands are accepted by all programs.

Selected Frame Buffer Tools	
fb-pix	frame buffer to color image
fb-bw	frame buffer to black and white
fb-cmap	read a frame buffer color map
fbcmap	can load several "standard" color maps
fbclear	clear to an optional RGB color
fbgamma	load or apply gamma correcting color maps
fbzoom	general zoom and pan routine
fbpoint	select pixel coordinates
fblabel	put a label on an image
fbcolor	a color selecting tool
fbscanplot	scanline RGB intensity plotter
fbanim	a "postage-stamp" animator
fbcmrot	a color map rotator
fbed	a frame buffer image editor

5.4. Image Manipulation

A collection of tools for image manipulation are provided. These can generate statistics, histograms, extract parts of an image, rotate, scale, and filter them, etc. Some of these are listed in the table.

Selected Image Tools	
pixstat	statistics - min, max, mean, etc.
pixhist	histogram
pixhist3d	RGB color space cube histogram
pixfilter	apply selected 3x3 filters
pixrect	extract a rectangle
pixrot	rotate, reverse, or invert
pixscale	scale up or down
pixdiff	compare two images
pixmerge	merge two/three images
pixtile	mosaic images together
gencolor	source a byte pattern
bwmod	apply expressions to each byte

6. User Interface

Using software tools effectively comes with experience. The BRL-CAD Package has tried to ease the difficulty of learning a new set of tools by using a common set of flags and common tool naming conventions throughout the package. The "user interface" is ultimately the Unix shell, and its conventions for establishing pipes, passing arguments to programs, etc. A shell with history recall and editing, such as the *tcsh*, is almost a necessity when constructing complicated command line pipes.

Constructing complex interconnections between processing tools from the command line is sometimes difficult. One limitation is the single input single output notion of a Unix pipe. Image manipulation often calls for three or more channels of data. The most common solution to this problem is the use of intermediate files. Other approaches include extensions to the *tee* program, or a special tool such as *chan*[12] which demultiplexes a stream, feeds each channel to a different program, and remultiplexes the results.

Recently several systems have been developed to facilitate the coupling of dataflow oriented tools. Stephen Willson of NRTC has developed what he calls a Layered User Interface.[13] This is a set of tools that provides generic buttons and sliders which can pass values on as tool arguments. Several of the BRL-CAD tools have been used in this environment. Dave Tristram of NASA Ames has put together a system called Flowtools[14] which allows the connections between tools to be specified with a dataflow like language, including inputs from sliders, etc. Both of these systems allow complex custom applications to be put together without writing any code.

7. Conclusions

The BRL-CAD Package is a Unix based system which provides a CSG solid model editor, a ray tracing library for model interrogation, a generic frame buffer library with network display capability, and a large collection of software tools. The library level interface to the ray tracer has allowed a large collection of model analysis tools to be incorporated into the system. The generic network capable frame buffer library has proven to be of tremendous day to day importance.

The package provides a flexible set of software tools for image manipulation. The image formats are extremely simplistic, something which has proven to have both good and bad characteristics. Approaches to providing higher level interfaces to tools of this form have been indicated.

References

1. MAGI, *A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles*, MAGI Report 6701, AD847576 (August 1967).
2. P. R. Stay, "The Definition and Raytracing of B-spline Objects in a Combinatorial Solid Geometric Modeling System," *USENIX: Proceeding of the Fourth Computer Graphics Workshop*, (Oct 1987).
3. M. J. Muuss, "Understanding the Preparation and Analysis of Solid Models," in *Techniques for Computer Graphics*, ed. D. F. Rogers, R. A. Earnshaw, Springer-Verlag (1987).
4. D. S. Kay, *Transparency, Refraction, and Ray Tracing for Computer Synthesized Images*, Cornell Univ (Jan 1979).
5. J. T. Whitted, "An Improved Illumination Model for Shaded Display," *Communications of the ACM* **23**(6) pp. 343-349 (June 1980).
6. M. J. Muuss, "RT and REMRT - Shared Memory Parallel and Network Distributed Ray-Tracing Programs," *USENIX: Proceeding of the Fourth Computer Graphics Workshop*, (Oct 1987).
7. D. F. Rogers, *Procedural Elements for Computer Graphics*, McGraw-Hill, New York (1985).
8. M. J. Muuss and P. Dykstra, K. Applin, G. Moss, E. Davisson, P. Stay, C. Kennedy, *Ballistic Research Laboratory CAD Package, Release 1.21*, BRL Internal Publication (June 1987).
9. Robert W. Scheifler and Jim Gettys, "The X Window System," *Transactions on Graphics* 5(2) pp. 79-109 (April 1986).
10. Tom Duff, "Compositing 3-D Rendered Images," *Computer Graphics* **19**(2)Proceedings of SIGGRAPH 85, (July, 1985).
11. J. W. Peterson, R. G. Bogart, and S. W. Thomas, "The Utah Raster Toolkit," *USENIX: Proceedings of the Third Computer Graphics Workshop*, (1986).
12. R. F. Moore, *CARL Startup Kit*, Computer Audio Research Laboratory, University of California at San Diego (1985).
13. S. Willson, "The Layered User Interface," *IRIS Universe*, (Fall 1987).
14. D. Tristram, "FlowTools: Dataflow Graphics Under Unix," *to appear, IEEE Conference on Workstations*, NASA Ames Research Center, ().

MEMORANDUM REPORT BRL-MR-3754

BRL

AN INTEGRATED ENVIRONMENT FOR ARMY, NAVY AND AIR FORCE TARGET DESCRIPTION SUPPORT

PAUL H. DEITZ
WILLIAM H. MERMAGEN, Jr.
PAUL R. STAY

MAY 1989

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

I. INTRODUCTION

For more than 40 years the vulnerability community has been developing analytic methods to predict the potential damage to targets from threats likely to be encountered in hostile engagements. Early methods of analysis consisted of manual calculations for bullet/target interactions. From the beginning, target geometry and material specifications were required input to the calculations. Such inputs were manually derived through use of blueprints and other system data. By the 1960's, the first attempts were made to apply machine-processing methods to problems of vulnerability assessment.

Two similar, but distinct, methods arose in the vulnerability community. One called the COM-GEOM/GIFT technique was developed by MAGI [1-4] for use by the US Army; the other called PATCH/FASTGEN was developed by FALCON [5] (now Denver Research Institute [DRI]). However the method of approach for each was identical. To perform a vulnerability analysis 1] a target description had to be generated. This file represented the three-space definition of geometry and coupled material information. 2] Mathematical rays, simulating bullet trajectories, were then passed through the target descriptions in order to find points-of-intersection, surface normals, line-of-sight thicknesses, and materials. 3] All of the information from step 2] was then passed to a vulnerability analysis were penetration relations and component-damage criteria were applied to calculate average system-level damage.

The difference between the COM-GEOM/GIFT and the PATCH/FASTGEN methods was principally in the schemes used to represent geometry. In the case of COM-GEOM, a set of simple shapes (called primitives) was defined, including four to eight-sided planar enclosures, an ellipsoid, a general cylinder, a general conic section, a torus, and a (constrained) faceted, self-closed shape used to model compound surfaces (such as cast turrets and aircraft bodies).

By contrast, the PATCH data base consisted uniformly of an (unconstrained) faceted representation. In each approach, the target description was developed by hand, with few automatic aides, and with none of the computer-aided design (CAD) assists that are familiar to all today. The bullet trajectory/material information was extracted from a COM-GEOM file via the GIFT code [6] and from a PATCH file via the FASTGEN code [5].

Over a period spanning more than fifteen years, two significant communities of vulnerability workers have developed generally using one or the other method. Because of incompatibilities between the geometric data structures (COM-GEOM vice PATCH), resources of one community were expended to replicate geometry already pre-existing in the other because of the incompatibilities of the techniques. Also, it can be assumed, useful analyses were foregone because existing target descriptions were incompatible and resources/time were insufficient to perform a duplication.

Against this background the BRL initiated some nine years ago a program with a goal to introduce modern methods of interactive computing to the problem of geometry generation, modification and interrogation. The initial task was to build an interactive editing environment so that BRL COM-GEOM descriptions no longer had to be built by hand. Later new ray casting utilities were generated to replace the old GIFT program used for many years. Also many image-handling and geometric data manipulation utilities were written to perform useful tasks. This suite of programs has become known as BRL-CAD [7] and consists now of some 150,000 lines of source code. This package has been distributed to over 300 computer sites around the world and supports a significant number of vulnerability, signature, and structural-analysis programs.

The aim of this paper is to summarize the chief properties of the BRL-CAD package, and particularly update current efforts to extend the old COM-GEOM data base to include both spline surfaces as well as the PATCH data base of DRI. This last extension represents a significant development for the US vulnerability community for finally both dominant target description techniques are supported within one consistent environment. With these advances, COM-GEOM, splines, and PATCH geometries are fully supported both for editing and shotline interrogation. Further, the various geometries can be used in "pure" form or in a "mixed" mode.

II. OVERVIEW OF BRL-CAD

The BRL-CAD package is written exclusively in C-code using structured coding methods. Individual processing modules are designed to support specific capabilities. Copious use is made of the "library" concept of software organization. By this method, software modules of general utility to a group of users are installed as system utilities and called by other programs under the control of the users themselves. This makes the individually tailored user programs much simpler, reduces greatly the volume of code maintenance, and allows utility upgrades and bug fixes to be available to all in an automated fashion. BRL-CAD is comprised of some 70 individual programs. Some of the principal elements are:

- *mged: Standing for Multiple-device Graphics Editor [7-9], mged is an* interactive editor for constructing and updating target descriptions of the COM-GEOM variety. Support is there for editing the basic six primitive shapes of COM-GEOM. Screen prompting is via a wire-frame representation of the solid-model data base. Many user aides have been added over the years including the ability to switch instantly between English and metric units, calculating volumes, surface areas, and (armor) fall-back angles for various shapes, region overlap checking; also commands to assign material property to regions, save a view for ray-tracing, and many others. The

display-oriented approach to building the COM-GEOM data bases is very easy to learn and can be used to build complicated models in a short time. Mged allows several graphics display devices to be used by the community for building COM-GEOM data bases. Some of the graphics devices include the Megatek 7000 vector display, Silicon Graphics Iris and 4D graphics workstations, Sun workstations, Tektronix vector displays, and Evans and Sutherland PS 300 graphics terminals. Recently added was a driver for the X Window System, which is used by many graphics display workstations vendors as a common graphics interface. Porting mged to new graphics devices is a straight-forward task and takes a short time to accomplish.

- *librt:* A library of functions suitable for ray tracing a target description file.
- *rt:* A lighting model whose input is based on ray tracing *(via librt* support). Up to ten light sources can be simulated, and objects can be given the properties of mixed diffuse and specular scattering, and refraction.
- *libfb:* A generic frame-buffer library which includes support for a number of display devices, as well as file, network, and debugging interfaces.
- *libplot3:* A public-domain version of the UNIX-Plot library has been extended with the following features; three-dimensional plotting support, 24-bit RGB color values, and floating-point values. These values are written into the plot file in a transportable, machine-independent binary format and can be used between different machines connected *via* a network.
- *util:* A collection of image-handling utilities, each constructed as individual tools intended to be used in combination. Such functions as color correction, format conversion, pixel comparison, and image filtering/processing are supported.
- *rfbd:* A "message-passing" interface layered on top of standard UNIX network protocols which allows image data to be transmitted from one computer and displayed on another.

BRL-CAD is designed to run under the UNIX™ operating system. This strategy has paid significant dividends by easing the porting of this code over many different computing environments and/or display devices. Some of the computers on which this software runs include:

- DEC VAX-11/750, VAX-11/780, VAX-11/785, VAX-11/8600, & VAXSTATION II GPX
- GOULD PN6000 and PN9000 Series
- Sun Workstations
- ISI 68020
- Ridge 330
- Pyramid 90Mx
- Elxsi 6400 Series
- Convex C1 XP & C2
- Multiflow TRACE 7/200
- Alliant FX/8 & FX/80
- Silicon Graphics IRIS 3000 Series & 4D Workstations
- CRAY X-MP/48 & CRAY 2

III. BRL-CAD GEOMETRIC DATA REPRESENTATION

The heart of a geometric modeling system is reflected in the data representation for three-space geometry. As noted above, BRL-CAD was originally designed to support the six classes of primitives under the old COM-GEOM file structure and to be interrogated by the old ray casting program GIFT [6]. These primitive types are illustrated in Fig. 1. The advantage of geometric description by such primitives is that the mathematical description is generally economic. In the case of the ARBs, the corners are specified. In the case of the second-order primitives (conics, ellipsoids) and the fourth-order primitive (torus), the descriptions are economic too and the surfaces are mathematically smooth. Their economy is also their limitation, in that the degrees-of-freedom of each primitive shape are clearly *limited*.

In the old COM-GEOM file structure, all of the primitive objects were simply numbered in ascending order. Only numerical designations could be given, so interpretation of objects by name was difficult. The current MGED file structure is hierarchical so that a target description can be constructed in multiple levels of logical groupings. English names can also be assigned. An instancing feature also has been added so that a single object prototype (round of ammunition, vehicle wheel, etc.) can be replicated to multiple positions and orientations in space. By this strategy, if the prototype is changed, all copies change automatically.

For certain classes of analysis, critical errors result if complex geometries are approximated by faceted representations (such as the ARS). In order to accommodate high-precision surface modeling, the MGED data base was extended some years ago to support the Non-Uniform Rational B-Spline (NURB) representation of the Alpha_1 geometry system [10-13]. An example of a part modeled with Alpha_1 is shown in Fig. 2. The strategy followed by the BRL has been to use the Alpha_1 editor, SHAPE_EDIT, to perform the actual construction of spline entities. MGED is then used to read the spline geometry into the target space, supported by global rotate, translate, and scaling capabilities.

A recent extension of the MGED geometry environment has been the inclusion of the PATCH representation used by DRI. The approach used here is to represent all shapes by triangular (patch) regions. For

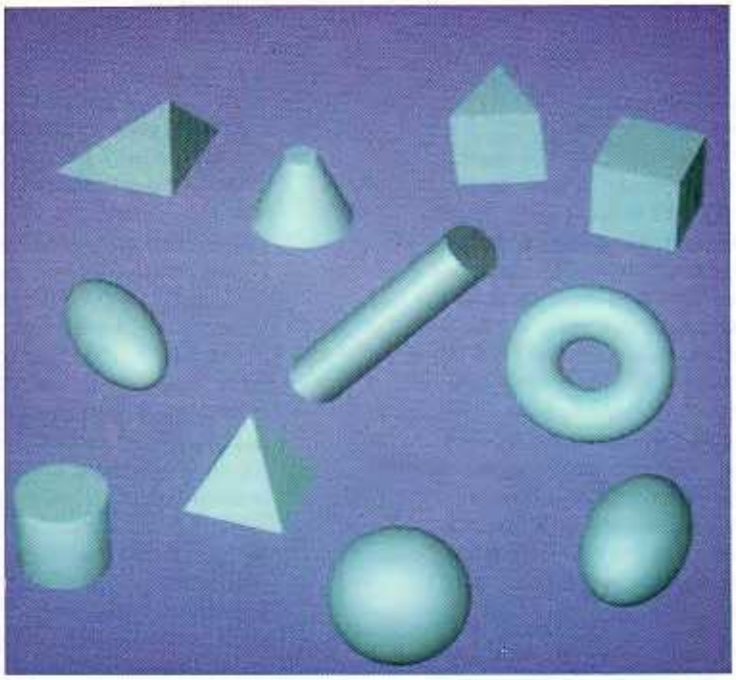


Figure 1. The set of primitive objects originally supported by the BRL-CAD environment. Basic shapes include the ARBs (four- to eight-sided planar shapes), the ellipsoid, a general cylinder, a general conic section, a torus and a (constrained) faceted object called the arbitrary surface (ARS).

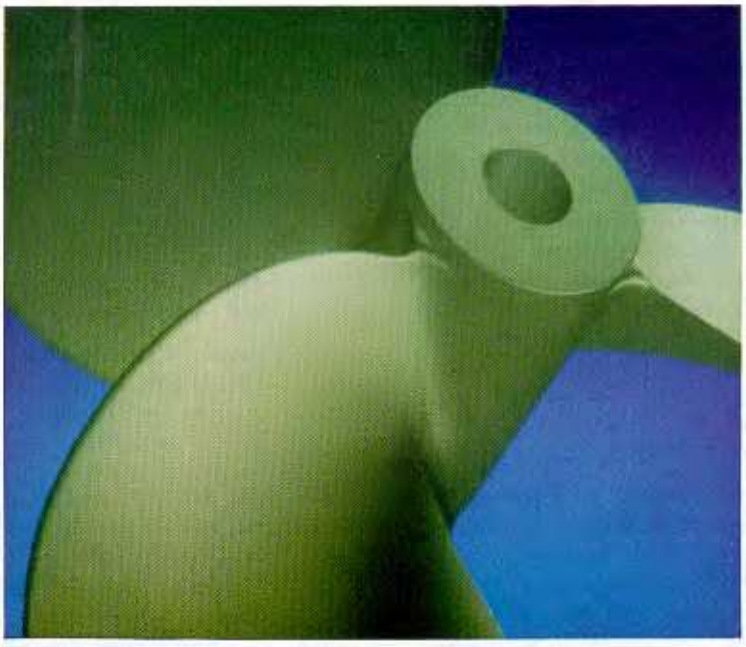


Figure 2. A geometric description of a ship screw built using spline geometry supported by the U. of Utah system, Alpha_1. Such spline geometry can be merged into the BRL-CAD data base. The use of splines to model complex surfaces avoids the use of faceted approximations. For certain applications where high surface fidelity is required, this capability is critical. *(Geometry courtesy of the U. of Utah.)*

flat surfaces, the approximation is exact. For curved surfaces, the degree of approximation depends on the number of facets used to represent the geometry. Figures 3a and 3b show an example of a DRI-created description of an F-14 aircraft. The wireframe image (Fig. 3a) shows the visual support given a target describer while in MGED. Figure 3b shows the rendered version of that aircraft created by the *rt* lighting model program. It is significant to note that the lighting model program was written well before either the PATCH or spline geometries were added to the BRL-CAD environment. Accommodation to interrogate new geometries is achieved entirely at the ray casting process supported in the library function *librt* to be discussed in the next section. However such extensions *do not* change the formatting of data passed from the ray interrogation process, so that all application codes whether a lighting model, as here, or a vulnerability application need undergo no changes whatsoever.

The inhomogeneous data base strategy described above has a number of important features and capabilities:

- Target descriptions for the purpose of vulnerability analysis (or any other application) can now be assembled and interrogated in the single MGED environment. If the geometry in itself is sufficiently accurate, there is no need to perform a "translation" from one representation to another. The approach here is simply to merge, as is, all geometric representations into a single environment. Thus a target description can consist as only PATCH, only spline, only COM-GEOM shapes, or any mixture of the three.
- For some applications, particularly in the area of signatures, faceted geometry (either through use of the ARS or PATCH shapes) embodies surface approximations which introduce serious errors in subsequent analysis. The inclusion of splines, together with surface fitting routines, gives an efficient upgrade path when needed. This capability is illustrated in Fig. 4 in which a US tank turret is shown first in a faceted representation and then transformed into a smooth-body spline shape.
- On the other hand, spline geometry is expensive in terms of data storage and interrogation. For high-detail point-burst vulnerability analysis, many thousands of interior vehicle or aircraft parts must be described. The relatively simple shapes of COM-GEOM adequately reflect the presented area and spatial position of these components without heavily taxing computer storage or processing requirements.

IV. LIBRARY SUPPORT

As noted in Section II, BRL-CAD makes copious use of the library approach to software support. Library routines are executable codes which are installed as part of the standard set code for use by all computer users. A number of the more important library routines are summarized:

- *librt:* Several routines which compute the geometric ray/intersection calculations with all primitives in the data base have been included in librt. By using these common routines, the applications programmer can place emphasis on modeling and programming the physics of a problem rather than invest time to recode the geometric properties of the model. The application sends the data base and its associated sub-trees to the library. A ray origin and direction are specified; other options include stopping the ray after first, next, or all object intersections. The library then performs the following calculations: 1) space partitioning, 2) bounding volume calculations, 3) ray/geometry intersections, and 4) boolean evaluations, and performs both parallel and vector execution, if available. The library then can return the following information to the application if needed: 1) hit points, 2) distance from the ray origin, 3) surface normals, 4) material property, and 5) surface curvature. The library is designed such that new primitives can be added easily to the system. Using this feature both NURBS [14] and PATCH models were added to the BRL-CAD system.
- *libfb:* The framebuffer library provides device-independent access to raster displays for displaying a common image format consisting of pixels of red, green, and blue values. There are several device drivers to support a variety of displays including Adage Ikonas, Silicon Graphics IRIS 3030 and 4D, Sun, and AT&T 5620 terminals. There is also a debug and disk interface. The library is designed to provide isolation from the low-level functions necessary to interact with the display hardware. This library uses the *libpkg* library to support display devices connected to other machines on the network. Library routines supporting the following operations are available for displaying images: open, close, zoom, pan, read/write pixels, read/write color maps, and cursor manipulation.
- *libwdb:* The libwdb library permits writing of MGED data bases from arbitrary programs. Though it does not currently have the full spectrum of MGED primitives, it does include the following primitives: half-space, rectangular- parallelepiped, arb4, arb8, sphere, ellipse, torus, right circular cylinder, truncated right cylinder, spline, and facet. In addition, this library permits the creation of "regions" and manipulation of their material property parameters.

  This new capability has made conversion from other geometrical data base types to MGED easier to implement and hence more efficient. In particular, the conversion code that converts PATCH descriptions into MGED data bases makes significant use of the libwdb library routines.

V. OTHER INTERFACES TO APPLICATIONS CODES

As noted in Section IV, ray casting in BRL-CAD is supported *via* the *librt routines. Ray casting has always*

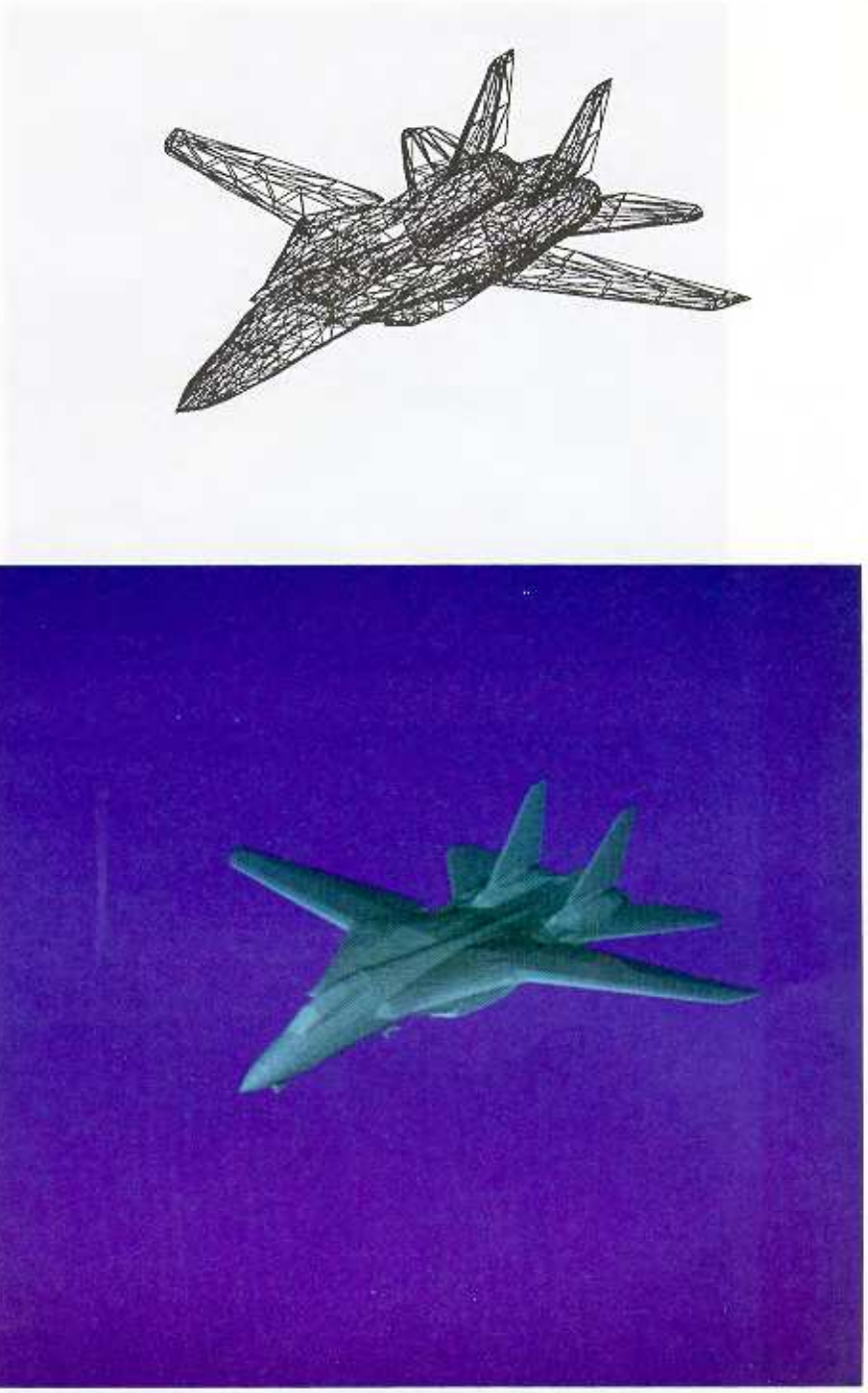


Figure 3. Part of a F-14 target description built using PATCH geometry due to DRI. The geometry on the top is shown as it appears when displayed by the graphics editor, MGED. The wireframes bounding triangular regions delineate the edges of the individual patches. The image on the bottom shows the object following rendering with the BRL *rt* lighting model. *(Geometry courtesy of DRI.)*

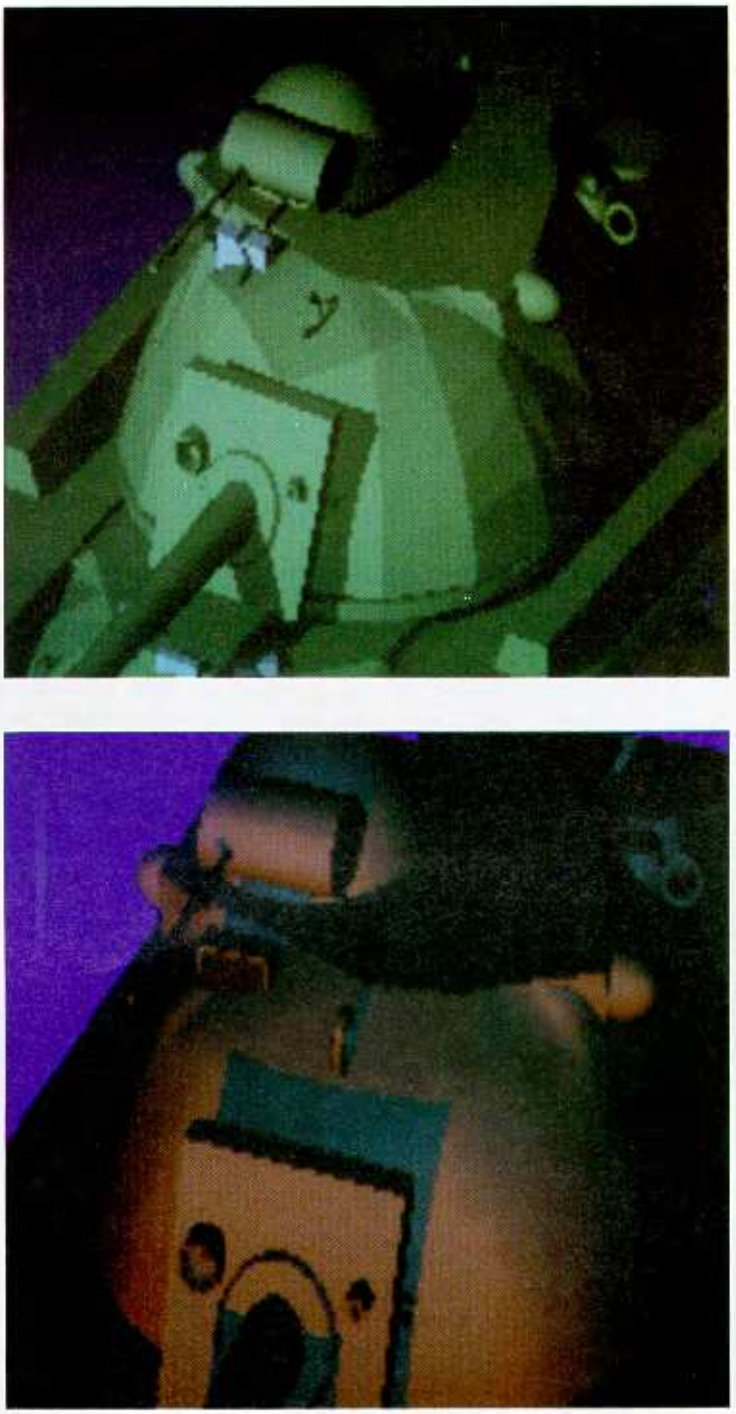


Figure 4. Images of an M48 tank built originally with a faceted turret on the top. On the bottom is illustrated the same turret after upgrading through the use of a spline fit. *(Geometric model by P. Stay, BRL.)*

been central to vulnerability analysis in order to simulate bullet trajectories. It is also used by many other applications codes including lighting models and various programs to predict signatures.

It is important to note, however, that there are other approaches to linking geometry to applications. These have been discussed elsewhere [15], but are summarized briefly here:

- Topology: Certain radar codes use a representation of geometry consisting of various canonical shapes such as flat plates, dihedral, and trihedral surfaces. Utilities have been generated which can be used to process a MGED target description to extract these shapes *via* processing filters.
- 3-D Surface & Volume Meshes: Many important mechanical and structural codes (ADINA™, NASTRAN™, etc.) are supported by such mesh structures. In order to support meshes which are direct derivatives of MGED geometry, a commercial modeling system called PATRAN™ has been linked *via* a translation code [16]. This conversion program maps each of the COM-GEOM primitives into the corresponding representation in PATRAN. Then PATRAN is used to generate the desired mesh.
- Analytic Representation: Each of the COM-GEOM primitives can be mapped into splines; the same is true for any PATCH object. Splines are one of the few geometric representations which can be manipulated analytically. One such property is that splines can be Fourier transformed. A number of radar modeling groups have attempted to take advantage of this and related properties.[17]

VI. OTHER UTILITIES/CAPABILITIES OF BRL-CAD

- Lighting Models: Two ray tracing programs that use the ray casting support of *librt* are provided in the BRL-CAD package, *rt* and *lgt; lgt* is an optical rendering program with a screen-oriented user interface. *lgt* has the ability to provide animation scripts and laser target renderings. *rt* also provides rendered images with command line arguments, but is itself the front end for several applications including a radar model. *rt* also has the ability to read scripts of commands which can control the computation of a sequence of frames and the orientations and properties of materials in each frame of an animation.

  Figures 5 and 6 show exterior views of the Bradley Armored Fighting Vehicle rendered using *lgt*. The target description used here is extremely detailed so as to support a high-frequency radar simulation. Figure 7 illustrates a lighting model option in which armor is rendered transparent so that internal component placement can be viewed.

- Animation: The continuous control of viewing position and/or the changes in relative geometry are of growing importance both as a tool to understand geometry itself as well as key to many applications including the multi-spectral signature area. The control of animation is achieved in the following fashion. Using *mged* a few frame positions (keyframes) can be selected using the saveview command. After multiple keyframes are generated, the program *xlate* fits a spline curve to the saved frames and generates additional frames to create a smooth animation sequence. The *rmats* command within *mged* will read processed frames to inspect the wireframe version of the animation, which can then be used to generate a series of images that can be transferred to video tape.

Ray Tracing Benchmark Results		
Rays/Sec	VAX /780	Machine
107.7	0.96	VAX Station II GPX
112.1	1.00	VAX 11/780
119.0	1.06	SGI IRIS 3030
127.3	1.14	Sun 3/50
191.8	1.71	VAX 11/785
413.4	3.69	GOULD 9080
521.1	4.64	VAX 11/8600
571.0	5.09	Sun 4/260
982.8	8.76	SGI 4D/60T
3972.9	35.44	Alliant FX/8, 8 CEs,
5376.3	47.96	Alliant FX/80, 8 CEs
7275.8	64.90	Cray 2, 4 CPUs,
13320.2	118.82	Cray XMP/48, 4 CPUs,

- Benchmarking Support: Finally, as the capabilities and costs of computing change rapidly, it is important to understand the benefits and limitations of the growing number of candidate machines and display devices in the market place. This has been achieved in part by utilizing a number of standard benchmarks for processing standard target-description/lighting-model images. The above table shows the results of running the program *rt* with a common data base and comparing them with the speed of a Digital VAX 11/780.

VII. APPLICATIONS CODES WHICH INTERFACE TO BRL-CAD

There is a large number of applications codes which interface to BRL-CAD. They have been enumerated in some detail elsewhere [18], and are simply listed below:

- Weights and Moments-of-Inertia
- An array of Vulnerability/Lethality Codes
- Neutron Transport Code
- Optical Image Generation (including specular/ diffuse reflection, refraction, and multiple light sources, animation, interference)
- Bistatic laser target designation analysis
- A number of Infrared Signature Codes
- A number of Synthetic Aperture Radar Codes (including codes due to ERIM and Northrop)

Figure 5. Frontal view of the Bradley Armored Fighting Vehicle using the BRL *lgt* lighting model. The geometry, built with MGED and only the primitive shapes illustrated in Fig. 1, is highly detailed so as to support high-frequency radar simulations. *(Geometric model by K. Applin, BRL.)*

Figure 6. Rear view of the Bradley Armored Fighting Vehicle calculated as in Fig. 5. *(Geometric model by K. Applin, BRL.)*

Figure 7. Transparent rendering of the Bradley Armored Fighting Vehicle. Using the same target description file as in Figs. 5 and 6, a lighting model option allows armor to be rendered transparent, revealing internal component placement. *(Geometric model by K. Applin, BRL.)*

- Acoustic modal predictions
- High-Energy Laser Damage
- High-Power Microwave Damage
- Link to PATRAN™ and hence to ADINA™, EPIC-2™, NASTRAN™, etc. for structural/ stress analysis
- X-Ray Imagery

VIII. WORK IN PROGRESS/EXTENSIONS

There are a number of projects underway which will greatly extend the utility of the package. Some of these are:

- Translations to Facets: The flat-sided objects such as the ARBs, ARSes, and PATCHes, are, by definition, facetized. Objects which are represented by facets have important utility for certain types of display and data representation. In order to achieve the ability to generate homogeneous facetized geometry, algorithms are being investigated to convert each of the higher-order representations into a facetized approximation in which facet size is under user control. A more difficult issue is the resolution of overlapping primitives. In the shotline interrogation process, boolean definitions (UNION, INTERSECTION, DIFFERENCE) are used to logic process the geometric rays. The resolution of boolean precedence along a single line is a relatively simple operation. The boolean resolution of overlapping meshes is much more difficult.

  There are a number of important payoffs for extensions along these lines. A direct translation of the mixed MGED data base to facets would achieve the following objectives: 1) Support of codes mentioned above which require 3-D surface meshes would be direct. 2) Also, many modern display devices support real-time polygon fill capability via hardware. This provides for much more realistic object rendering than the usual interactive wireframe images.

- Automated Drafting: In previous years the BRL has seldom needed to generate standard blueprints from its solid geometry. Nevertheless, such a capability would be a useful extension. A program is being put in place to make this improvement *via* a small business contract.

- Data Base Extensibility: We note again that the current BRL-CAD geometric data base is distinctly non-homogeneous. When another modeling scheme has been used to represent geometry, and when it is important to utilize that geometry in the BRL-CAD environment, there are two paths to compatibility. The first is to see if the new geometry has an exact, corresponding representation with the current BRL-CAD primitives. If so, it is simply a data reformatting job to make a MGED-readable input file. However, if the new geometry does not correspond to any current structure (previous examples include both the spline and the PATCH data bases), then the current MGED data base is simply extended to include the desired representations. This requires work at two distinct points. The first is in the formatting of the MGED data base and the graphical imaging and manipulation tools in the MGED editor itself. The second point is in *librt;* there the ray casting tool must be modified so that it knows how to perform the intersection, surface normal, curvature, etc. calculations. However, once these changes are made, no other modifications are required, and all application codes run with no change.

IX. SUMMARY

In this paper we have described a unified set of software which brings the two major geometric data bases used for vulnerability analysis under a single integrated environment. Spline surfaces are supported as well, making available an important growth path when high-precision geometry is required for demanding analyses.

Because of these efforts, it is no longer necessary to duplicate identical target geometry because of two incompatible representations. In fact, mixed modes of targets can be assembled using arbitrary combinations of COM-GEOM, PATCH, and spline data bases.

This software has been designed to run on more than a dozen brands of vendor hardware. Machines from the SUN Workstations to the CRAY 2 are supported. Since the source code for all of this software is Government owned, it can be ported to any desired target machine without cost of royalty or suffering vendor constraint.

Finally, the code has been built in modular blocks. This makes for easier development and enhancements. It is to be expected that the evolution of this package will continue to reflect both user needs and the rapid development of higher speed machines and display devices.

REFERENCES

[1] "A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles," **MAGI-6701**, AD847576, August 1969.

[2] "The MAGIC-SAMC Target Analysis Technique," **Vol VI, AMSAA TR14,** April 1969.

[3] "MAGIC Computer Simulation", **Vol. 1, User Manual**, 61JTCG/ME-71-7-1, July 1971.

[4] "MAGIC Computer Simulation", **Vol. 2, Analysts Manual Parts 1 and 2**, 61JTCG/ME-71-7-2-2, May 1971.

[5] "FASTGEN 3 Target Description Computer Program", **Technical Report ASD-TR-85-5019**, Logicon, Inc. for Air Force Aeronautical Systems Division, Wright-Patterson AFB, Ohio, 10 December 1985.

[6] G. G. Kuehl, L. W. Bain Jr. and M. M. Reisinger, "The GIFT Code User Manual; Volume II, The Output Options (U)", **US Army ARRADCOM Report No. 02189** (AD #A078364) September 1979. GIFT stands for *G*eometric *I*nformation *F*or *T*argets.

[7] **Ballistic Research Laboratory CAD Package, Release 1.21 (2-June-1987)**, "A Solid Modeling System and Ray-Tracing Benchmark Distribution Package", SECAD/VLD Computing Consortium.

[8] M. J. Muuss, K. A. Applin, R. J. Suckling, G. S. Moss, E. P. Weaver and C. Stanley, "GED: An Interactive Solid Modeling System for Vulnerability Analysis", **BRL Technical Report ARBRL-TR-02480**, NTIS AD# A-126-657, March 1983.

[9] M. J. Muuss, "Understanding the Preparation and Analysis of Solid Models", in **Techniques for Computer Graphics**, Edited by D. A. Rogers and R. A. Earnshaw, Springer-Verlag, 1987.

[10] R. F. Riesenfeld, E. Cohen and T. Lyche, "Discrete B-Splines and Subdivision Techniques in Computer Graphics", **Computer Graphics and Image Processing, Vol. 14**, No. 2, October 1980, pp. 87-111.

[11] E. Cohen, "Mathematical Tools for a Modelers Workbench", **IEEE Computer Graphics and Applications**, October 1983.

[12] R. F. Riesenfeld, "A View of Spline-Based Solid Modeling", **Proceedings of Autofact 5, Computer and Automated Systems Association of SME**, 1983.

[13] **Alpha_1: User's Manual, Engineering Geometry Systems**, SLC, UT (1987).

[14] P. R. Stay, "The Definition and Raytracing of B-Spline Objects in a Combinatorial Solid Geometric Modeling System", **USENIX: Proceeding of the Fourth Computer Graphics Workshop**, October 1987, Boston MA.

[15] P. H. Deitz, "Predictive Signature Modeling via Solid Geometry at the BRL", **Proceedings of the Sixth KRC Symposium on Ground Vehicle Signatures**, Houghton, MI, 21-22 August 1984.

[16] The program is *gedpat*, and it is documented in the **VLD/VMB UNIX Supplementary Manual**, D. A. Gwyn, Editor, August 1987.

[17] Private communications with M. Sancer, Northrop Corporation, and J. F. Dammann, USA Harry Diamond Laboratories.

[18] P. H. Deitz, Computer-Aided Techniques for Survivability/ Lethality Modeling, **Proceedings of the ADPA Survivability and Vulnerability Symposium IX**, held 28-30 October, 1986, Naval Surface Weapons Center, White Oak, MD.

Keynote Address

Reflections From The BRL Experience

as seen by

Harry L. Reed, Jr.
Former Chief, SECAD, BRL

V1S05A00

I expect that almost everyone here shares in the belief that CAD is an extremely useful and powerful tool - yea indispensable in this age of engineering; so I won't try to sell you on CAD. Rather I'll talk about some anecdotal history, some of my thoughts that come from looking at the history of CAD at the BRL, and talk about what I think are some of the challenges to the applications of CAD. I'll follow this list of issues:

[1] USEFULNESS - is a particular area of endeavor better off for having CAD?

[2] RELATION TO REALITY - is the image true to reality? More true?

[3] STABILITY - do refinements and "improvements" produce converging results? Are general trends discernible? Can results from previous versions be reconciled?

[4] GLOBAL UNIFICATION - do we have the proper approach to the knowledge base issues to allow the computer to be the unifying and facilitating tool it can and should be?

[5] THE NEED FOR HUMOR - the world of Computer Science and CAD is often strange.

1. USEFULNESS

In BRL, the demand preceded and drove the technology (at least initially). A couple of decades ago, tanks were whittled not designed. Wooden mockups were made, where more room was needed (say for ammunition handling) wood was gouged out, wood was added to the outside to maintain the armor protection, and when the process was completed the mockup was measured and drawings were made. The value of computer technology in this area can best be attested to by the recent acquisition of a Cray-2 by the Tank Automotive Command.

To conduct vulnerability analyses a variety of simple but practical tricks were used. For example pieces of paper that represented the outline of a target and of the presented area of components of interest (e.g. the engine) target were cut out and weighed. The ratios of the weights gave such quantities as the probability that a hit on the target would result in a hit on the engine (of course penetration also had to be considered). Planimeters used with drawings were a more sophisticated approach. With the need for many targets, many aspects, many munitions, various types of damage, etc. making and measuring all these drawings became a very labor intensive business.

Vulnerability analysis got a leg up with the introduction of computerized geometry (we called it COMGEOM) in the late 1950's. It relied on the use of over-the-counter batch computers and the primitive output of a line printer. Still it was a giant step forward.

A little over ten years ago, BRL got into the interactive computing world with the introduction of UNIX based tools and the first pieces of BRLNET. In developing CAD at BRL, Mike Muuss, Paul Deitz, et al have met a demand. This was not a case of technology looking for an application; this was an application that was hungry for the technology.

Likewise, BRL's interest in SMART weapons produced a need to be able to model target signatures and to simulate the end game situation in which a sensor on an approaching projectile or missile uses information from the target signature to provide a hit. In the early 1970's, people in the BRL tried to develop models for the thermal signature of tanks. Success was limited largely by CPU power and the effort was dropped. Now optical, IR, and radar signature modeling all seem very possible and promising.

2. RELATION TO REALITY

Let me start by saying that the BRL's CAD program has been outstanding as a balanced effort among experimentation, computation, analysis, and validation. I believe that you'll get more of that flavor from much of what you'll hear in these two days. This is true in both the vulnerability and lethality area and in the target signature area.

It's most important to understand what our computer tells us, and to make sure that we use the best data and theory that engineering science can provide. Sometimes we're asked to "take our best shot" when we know that our ground is uncertain. We do it - and rightly so as long as all concerned understand the limitations and the risks.

It is also often the case that the technologies that support the analysis of different aspects of a problem have considerably different maturities. For example, one major portion of an analysis may be supported by huge amounts of computation, and another part that is equally as important may only be addressable by very crude empirical methods. One must be very careful to appreciate the limitations of the total result, but the detailed calculations may well be extremely valuable in their own right. And of course, there is always a temptation to give these intensive calculations a specially intimidating role in espousing a position.

Also, often, our ability to do engineering and science are outstripped by our ability to do computer graphics. In many areas we are far form having an authoritative ab initio model. This certainly does not mean that we cannot learn new and valuable things from using these models (on the contrary they are exceptionally valuable). However we must always make sure that we understand new results - whether they indeed are correct and why they come about.

A few of the engineers with whom I chatted recently used the expression "Air Brush Physics". When pressed to explain that term, they could only express an uneasy feeling that sometimes a lot of fine tuning of a particular view might go on at the display tube and that the result might either not be so good as it appeared or that the production of numerous similar cases might be prohibitive. I don't want to over play this concern except as I think that it points out the very great importance for the practitioners of CAD to work closely with their clientele to make sure that everyone concerned understands the limitations and the abilities of the engineering and of the computer technology.

3. STABILITY

One issue under stability is convergence. Simply enough, that is the question of whether results improve or even converge to a set of results with additional complexity and sophistication of the models.

For example, in the area of target modeling for SMART weapons, there is a need for both detailed target signature data and for some sort of "averages" which relate to total system performance during the course of a war. However, the average signature is dull (verging on useless); averages must be taken after the application of the (specific) nonlinear decision algorithm. The complications of various ambient conditions, natural and man-made, backgrounds, and etc can make the evaluation virtually impossible using highly detailed models. And, of course, it often remains to be shown that increasing the number of runs would provide a (correct) convergent result.

On the other hand, using a small number of detailed calculations raises what I'll call the continuity problem. That is, to be useful, a model should have the property that vanishingly small changes in input and configuration should produce vanishingly small changes in output. Is there an instability of detail i.e. as we create more detail (with the concomitant increase in computing needed per scene) do we require more views and statistical samples to produce a suitably continuous product?

4. GLOBAL UNIFICATION

I am thoroughly convinced that CAD and related technologies such as networked computing promise entirely new vistas in the Total System Design and Analysis. Truly from cradle to the grave the system will be created, modified. evaluated, and produced with a continuous application of related computer programs.

The CRAY-2 for BRL and the CRAY-2 for TACOM were sold with just this idea in mind. A tank could be created as a total system - not a vehicle that carries around armor, or an armored truck or etc. But as an integrated design in which the armor is an integral part of the structure. Also some preliminary analysis could be done with mobility models to help select among designs and to provide promising designs without cutting a lot of metal for the losing designs. Of course testing of hardware must occur, but these tests could be limited to winning concepts and could be more efficient and effective with the computer to help to provide hypotheses for testing.

Of course, with more than one phase of the design process involving more than one computer center etc., there is considerable potential for reinventing the Tower of Babel. The essence of the computer in this sort of application must involve (among other things) "putting it in once and only once" - to use one and only one computer master file as the current and authoritative source of data that is easily usable and available for all application programs and that can be reliable, easily, and rapidly updated to reflect proper changes through the process from development of concept to engineering design to the operation of digital machine tools.

I know that Paul Deitz would love to have everyone use the BRL CAD package. This would certainly be nice, and it would represent one form of unification. But more important, I think, is the need to work the data base problem. We must always maintain seminal data (whatever the fundamental representation of the system is). Those data must be readily available to any appropriate application program in a transparent manner and likewise the updating process must be accomplishable from a similarly transparent process. In my mind this problem in knowledge representation is the major problem in this area of using CAD and related computer technology in a global system context.

5. NEED FOR A SENSE OF HUMOR and a very open mind

Computer science folks are very creative people, and sometimes march to quite different drummers. And of course the technology is spread over a most diverse landscape from hard engineering to creative art. Many times, good ideas come from strange (to the military) places; sometimes this includes computer games which are an anathema to many of the managers and controllers of computers.

I've been a bit of a Dutch Uncle today. I must say that I've been proud to have been part of the BRL computer experience and while I was chief of SECAD to have been able to brag about the CAD accomplishment that were a joint effort between our division and the Vulnerability and Lethality division - (or more specifically between Mike Muuss' team and Paul Deitz's branch). It's a special pleasure for me to be part of this symposium, and I certainly hope that you find your visit rewarding and enjoyable.

A Road Map Through the BRL-CAD Package

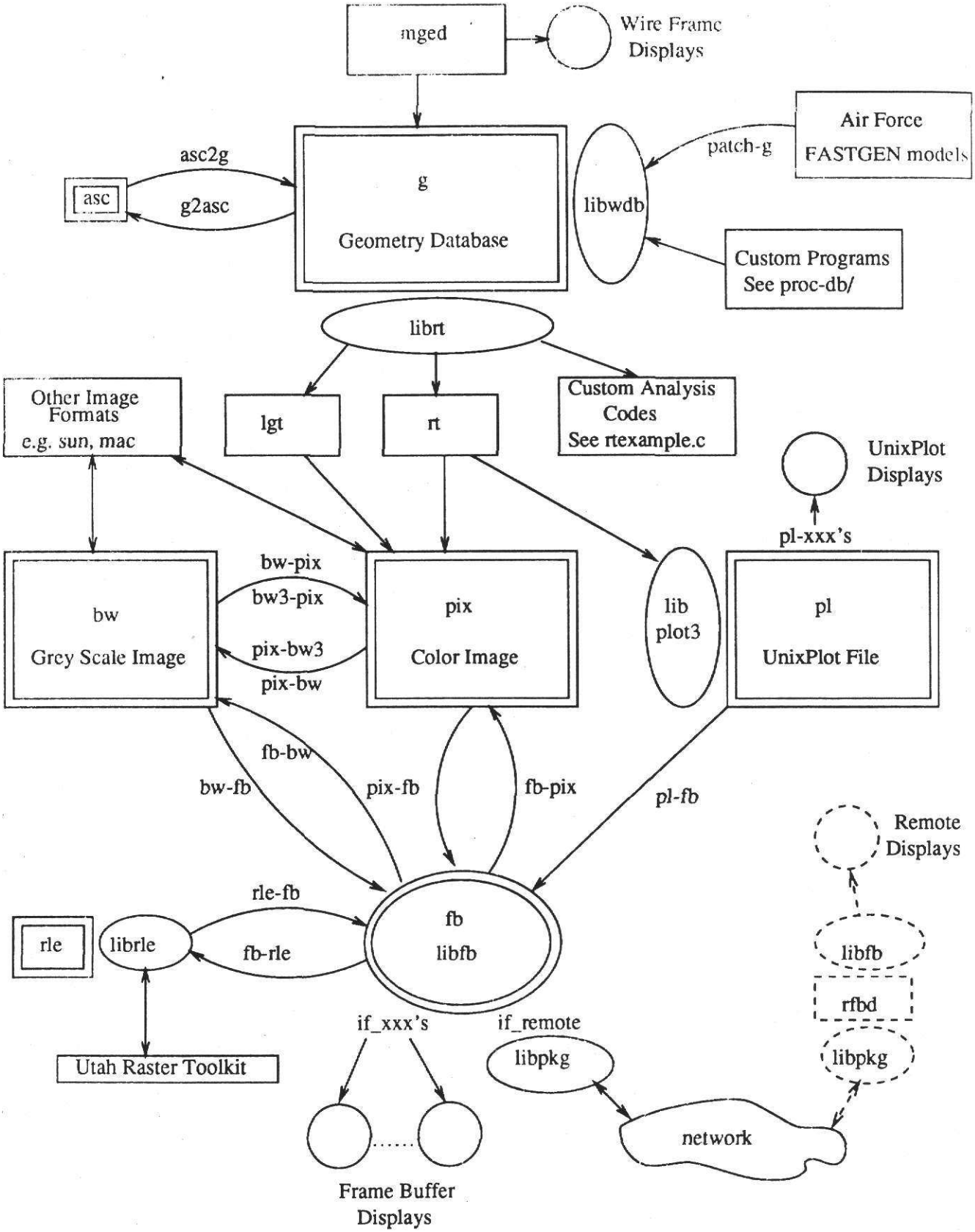
Phillip C. Dykstra

Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

ABSTRACT

This paper provides an overview of the various parts of the BRL-CAD Package and how they relate to each other. It does so from the viewpoint of the major data formats involved, and the **dataflow** between them, i.e. how these formats get generated, how that are modified, and what options exist to convert one form to another. Attention will focus on the big picture: the five or so most significant file formats, the most important creators and consumers of these formats, and the interfaces into and out of each format (e.g. the libraries involved). The goal of this paper is to make the overall picture accessible to the reader.

September 6, 1988


mged
Wire Frame Displays
Air Force FASTGEN models
patch-g
asc2g
g2asc
asc
g
Geometry Database
libwdb
Custom Programs See proc-db/
librt
Other Image Formats e.g. sun, mac
lgt
rt
Custom Analysis Codes See rtexample.c
UnixPlot Displays
pl-xxx's
bw-pix
bw3-pix
pix-bw3
pix-bw
bw
Grey Scale Image
pix
Color Image
lib plot3
pl
UnixPlot File
fb-bw
bw-fb
pix-fb
fb-pix
pl-fb
Remote Displays
rle-fb
fb-rle
rle
librle
fb
libfb
libfb
rfbd
libpkg
if_xxx's
if_remote
libpkg
Utah Raster Toolkit
network
Frame Buffer Displays

A Road Map Through the BRL-CAD Package

Phillip C. Dykstra

Advanced Computer Systems Team
U. S. Army Ballistic Research Laboratory
Aberdeen Proving Ground
Maryland 21005-5066 USA

1. Introduction

This paper provides an overview of the various parts of the BRL-CAD Package and how they relate to each other. It does so from the viewpoint of the major data formats involved, and the **dataflow** between them, i.e. how these formats get generated, how that are modified, and what options exist to convert one form to another. Attention will focus on the big picture: the five or so most significant file formats, the most important creators and consumers of these formats, and the interfaces into and out of each format (e.g. the libraries involved). The goal of this paper is to make the overall picture accessible to the reader. Details are found elsewhere.

2. The Big Picture

While the BRL-CAD Package can be described in terms of its major parts,[1] another perhaps more "holistic" view is as a large collection of interworking tools. The user is presented with a large body of software which can serve as a tremendous aide in accomplishing a diversity of tasks. It is perhaps even more powerful to the programmer who, by taking advantage of the many library or file format interfaces that exist, can link in any application specific pieces that may not be available in the general tools.

The figure in the front of this document shows a truly big picture of the BRL-CAD system from the viewport of dataflow. File formats are shown in double boxes, libraries in ellipses, and programs in single boxes. Programs that convert one format to another are shown on the arcs connecting those formats. Circles depict display devices.

The four most important file formats are shown. Also depicted is the frame buffer display interface, which is so significant to the BRL-CAD system, that it will be discussed in this paper as if it were another file format. While the picture may appear quite complicated, it is a quite simplified view of the total system. Many programs and connections were omitted for clearity.

3. Universal Naming Conventions

The short names in the boxes are the strings or file name extensions associated with these formats throughout the system. For example, "pix" is a 24-bit per pixel color image. These image files are usually named "file.pix", while programs that read or write them will have "pix" in their names. If a program converts one kind of data to another it will have a hyphen between the incoming and outgoing formats or destinations: e.g. "pix-bw" converts pix file input to bw file output. "bw-fb" takes bw file input and places it an a frame buffer (display device).* If a program operates on a single kind of data, or is the source or sink of data, its name will be prefixed by that format. Examples: "fbzoom" performs panning and zooming on a frame buffer; "pixmod" modifies values in a pix file.

As one of the stranger examples in the system, there exists a program called "fb-cmap" which

---

A couple of old exceptions to this rule are g2asc, pix2asc, and their inverses, where a "2" was used instead of a hyphen.

takes a color map from a frame buffer and outputs it to a file, as well "fbcmap" which places one of several built-in color maps on a frame buffer (it sources the data, and thus no hyphen, and operates on a frame buffer, and thus the prefix fb), and finally there is the program "cmap-fb" which takes a color map file as input, and places on a frame buffer. [There is no cmapfb. Think about it - such a name would imply that it performs an "fb" operation on a color map file, whatever that might be.] If the reader understands this last example, you are well on your way toward understanding the entire system!

The major formats:

g - geometry data base.
pix - 24 bit per pixel color image.
bw - 8 bit per pixel black and white image.
pl - UnixPlot simple plot file description.
fb - frame buffer display interface.

See the Appendix for a more complete list.

4. g - Geometry Database File

This is a binary Constructive Solid Geometry (CSG) database of a model. You will sometimes see the geometry description that it contains called COMGEOM. It consists of geometric primitives (ellipsoids, cones, spline solids, etc.), a hierarchy of boolean combinations of these (union, intersections, subtraction), and a certain amount of material type identifiers and so forth. The file h/db.h defines the exact structure of this database.

Of all the major formats discussed, this is the only one that is machine dependent in that it contains machine specific integer and floating point values (a future release will change to a machine independent one). To move a database from one machine to another it must be converted to Ascii text on the source machine (g2asc) and then back to binary on the destination machine (asc2g).

There are three ways in and out of a g file:

1) libwdb provides a means for programs to create/write databases.
2) librt has some routines for reading, but most importantly for tracing rays through the geometry in a g file.
3) directly reading and writing the structures.

The major producer of these files, and also one of the largest single components of the BRL-CAD system, is the solid model editor MGED (Multi-device Graphics EDitor). MGED interacts with the user through one of several "display managers," which provide a "wire frame" interface to a display device. One display manager exists for each kind out output device supported. The display managers are all compiled into MGED and are found in mged/dm-XXX.c. A future release will break these out into a library - libdm.

Another way of creating these g files are from programs which use libwdb (see for example the directory proc-db), or converters from other solid model databases (e.g. the Air Force Patch format converter).

The major consumer of g files are analysis codes that interrogate the geometry via librt. As a simple programming example of using this library see librt/rtexample.c. Shown are the two rendering programs, lgt and rt, both of which produce pix file images. Others include rtshot, rtray, rtrad, etc.

RT is itself a multiheaded program. It has an internal interface to "view modules" which control what happens at each ray/geometry intersection, and what kind of output gets produced (see rt/view*). Several different programs are produced by linking the common front end with different view modules.

5. pix - 24-bit Color Image Files

The major image format used in the BRL-CAD Package is a 24 bit per pixel RGB color file. It has the simplest structure possible - no header whatsoever, just 3 8-bit unsigned values for each pixel representing intensities from 0 (min) through 255 (max) for the Red, Green, and Blue channels. These pixels are stored in first quadrant order as RGBRGBRGB... from lower left, across the rows left to right, and up the image from bottom to top. See [1] for why we chose this.

Because this format is so simple, there is no library interface to these files. Instead programs simply read and write bytes directly (usually with the Standard I/O fread/fwrite calls). Largely because of this simplicity (and generality - often the data manipulated need not be an image at all), we choose this format for the basis of most of our image manipulation and processing tools. Rendering programs such as rt and lgt produce these, and other image formats can be imported and exported to and from pix files for processing and display. A very large number of programs exist for dealing with pix files themselves. To name a few examples these could be grouped roughly into information (pixstat, pixhist), modification (pixmod, pixfilter), manipulation (pixrot, pixcrop), comparison (pixdiff), composition (pixmerge). [The frame buffer also plays a major role in the processing and composition of images.]

One of the more notable formats that pix files can be converted to and from are rle files. These are the University of Utah's Run Length Encoded compressed files. The BRL-CAD package treats rle files primarily as a compressed storage format only, and while we provide a means of displaying these files directly, all processing is done by converting them to the much simpler pix format. An alternative approach that the University of Utah has taken in the Utah Raster Toolkit (included in the BRL-CAD release) is to provide a library interface to the rle files and center all of the utilities around this format, by having them all use that library.

6. bw - 8-bit Black and White Image Format

The bw file format is exactly like the pix format except that there is only one byte per pixel, usually assumed to represent a shade of gray from 0 (black) through 255 (white). A large collection of tools exists for bw files as well, most often the exact same utilities as the pix ones except with a bw instead of a pix prefix. [It is worth noting that a few of the pix programs are generalized to handle "pixels" of not just three, but any number of bytes via a -#<bytes> flag. This allows many clever trick to be performed. For example matrices of floating point values can be transposed with pixrot by specifying the data size and matrix dimensions!]

Sometimes it is more convenient to process color images as individual color planes. pix files can be converted to either three bw files, one per channel (pix-bw3), or the planes can be smashed together in a variety of ways (pix-bw). The reverse programs are also provided.

7. pl - UnixPlot Files

Both Berkeley BSD Unix and AT&T System V Unix systems usually include a simple device independent plot file format plot(5). This format is very simple, consisting of moves and draws, points, circles, simple text, etc. It is machine independent, consisting of Vax order 16-bit signed integers. Filter are included called tXXX to output these files on various devices (e.g. t4014).

BRL has extended this simple file format to include three dimensional points and lines, and machine independent floating point (64-bit IEEE). A public domain libplot(3) replacement called libplot3(3) is included that allows programs to write this extended format. Floating point plotting has proven dramatically superior to arbitrarily scaled integer plotting spaces in that natural coordinates can be used (allowing different plots to be naturally combined and coordinates read from the plot file itself), and the plots can be dynamically scaled over several orders of magnitude without loss of precision.

Pl files can be viewed in Ascii for debugging (pldebug), rotated, translated, or scaled (plrot), converted to the simple 2D integer format for consumption be the standard Unix utilities (pl-pl), or viewed on a number of devices (e.g. pl-fb, pl-sgi, pl-X).

Other links to pl format are that rt can produce pl plots of ray paths and geometry

bounding boxes, and MGED can overlay a plot file over the existing geometry display (used with rtoverlap for overlay checking).

8. fb - Frame Buffer Library

The frame buffer library and tools are where many operations occur. Simple compositing and comparison of images can be done, tiny animations can be run in real time using pan and zoom, color maps can be altered for image detail/contrast enhancement, etc. Each type of hardware display supported has an module in libfb called if_DEVICE.c. Any program compiled with this library is now able to directly control any of the display types supported in that library. Several default interfaces are always available including a debug interface and a "stacker" that allows multiple displays to be ganged together.

Of particular note is the remote network interface. This interface passes all libfb calls across the network to a daemon on a remote machine. These called are then passed on to any of the displays on that machine via rfbd (the Remote Frame Buffer Daemon). The power and utility of this can not be overstated. It frees one of the distinction between where they might be sitting (e.g. what displays are available near by) and what computer they are using. It also gives sight to ordinarily blind machines such as the Cray supercomputers.

9. Other Programs

Not shown on the diagram are a very large number of programs that operate on the file formats directly. Most notably, the pix, bw, and pl file formats. See that manual pages for these. Other libraries that programs in the package may use include:

libtermio - BSD and System V independent terminal support
libsysv - System V compatibility routines
libspl - spline routines, usually included via librt
libfont - vfont access routines

10. Conclusion - The User

Where does the user sit in all of these pictures? Typing to the Unix shell, the user has the ability to execute any of the programs on the arcs between these different formats, as well as all of the programs that generate or manipulate a single format. This places you in the role of the conductor, where you orchestrate how all of these tools get linked together to perform the desired task. As a programmer, you can also utilized any of the library interfaces or file formats to couple with this existing body of software extremely easily.

Appendix - Abbreviations found in the BRL-CAD software.

ap - Applicon color ink jet printer.
asc - any one of several "printable Ascii" formats
(usually to avoid binary for portability).
bw - 8 bit per pixel black and white image.
cat - C.A.T. phototypesetter code ("standard" troff produces this)
ci - SGI "Color Image" format
cmap - a frame buffer color map file
dit - Device Independent Troff code (Documenters Workbench troff)
dvi - DeVice Independent code (TeX output)
fb - frame buffer display interface.
g - geometry data base.
imp - Imagen IMPress page description language
mac - Macintosh bitmap.
op - Optronics film scanner.
pix - 24 bit per pixel color image.
pl - UnixPlot simple plot file description.
pp - "pretty picture" output from an old ray tracer called GIFT.
ps - PostScript
rad - a ray history file used by the Radsim radar simulator.
ray - "VLD Standard" ray file format (don't ask).
rle - University of Utah Run Length Encoded (compressed) image.
sgi - Silicon Graphics Inc. workstation.
sun - Sun workstation bitmap (or the workstation itself).
X - the X Window System.

References

1. P. C. Dykstra and M. J. Muuss, "The BRL CAD Package: An Overview," *USENIX: Proceeding of the Fourth Computer Graphics Workshop*, (Oct 1987).

distributed graphics for high-resolution 3D modeling

By Michael John Muuss

Ballistic Research Laboratory's CAD Package, now listed in Silicon Graphics' Software Exchange program, consists of three major elements, the most significant of which is a solids-modeling system (complete with a ray-tracing library). The system's second important element is a set of libraries for producing graphical output on varied display hardware. Comprising the third component are numerous software tools, each of which is focused on a single function but designed to be combined via UNIX piping with other tools. The technology and software employed in shaping these pieces of the CAD environment are described in some detail in "Understanding the Preparation and Analysis of Solid Models" [*Muuss87*].

The intent of Ballistic Research Laboratory (BRL) in designing the CAD package was to enable the creation and analysis of highly detailed three-dimensional solid models. The strategy of the package's designers was to develop a broad set of analysis tools supported by the same geometry database. Using these tools, the strength, weight, protection, and performance levels offered by structures in a model can be readily assessed. As a result, highly detailed designs can be constructed using a philosophy of *system optimization* right from the start, thus allowing for the rapid development of designs offering desired performance at an appropriate price.

analysis tools

Users of the BRL CAD Package are provided a powerful platform upon which to build analysis tools, where the physics of a particular analysis are insulated from the complexities of the shapes under analysis. Analysis tools are built, free from the details of an underlying geometry, using the package's ray-tracing library for model interrogation, **librt**. Ray-tracing makes it easy to deal with objects that are either partly or entirely reflective; the same is true of transparent objects of varying refractive indices, which--like highly reflective objects--would otherwise be difficult to model. By applying proper dither, motion-blur, depth-of-field, translucency, and shadow penumbra, other effects can be achieved. The package, morover, makes it easy to animate the position of the "eye" (camera) and light sources and to articulate the model geometry itself. Both operations can prove quite beneficial in comprehending complex geometry.

By recording light ray paths, one may follow light that's refracting through lenses and reflecting from mirrors while rendering an image. Lens behavior can be predicted, including exact focal length, influences of spherical distortions and edge effects, amounts of image distortion due to internal reflection and scattering, and levels of reflection due to lens-mounting hardware. Experiments, moreover, can be conducted to determine the effects of adding or removing baffles, irises, special lens coatings, and the like.

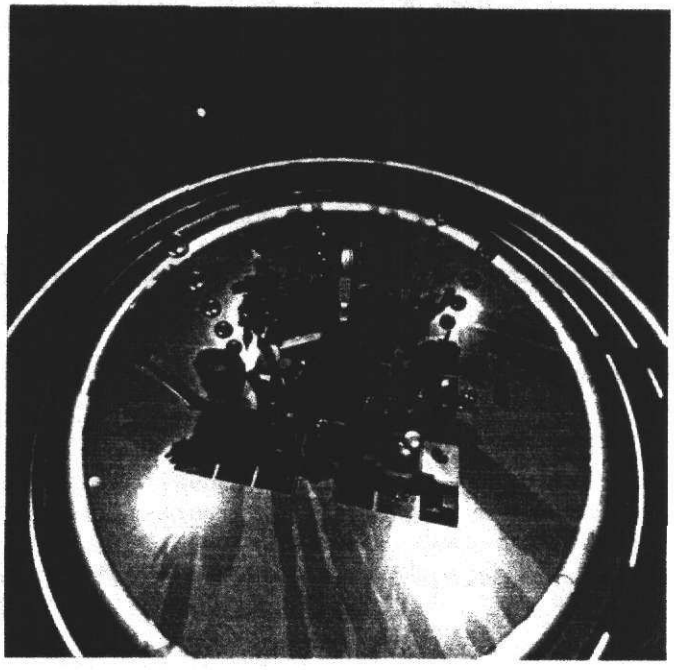
Another analytic problem that BRL's CAD Package provides for has to do with moments and products of inertia. Particularly when designing aircraft, weights, gravitational centers, and parameters related to inertia are vital issues to consider in creating stable, high-performance designs. Moments of inertia are also important in determining the conditions under which a vehicle might be overturned; they, for instance, can help in predicting the vehicle's handling during maneuvers over rough terrain.

Synthetic aperture radar (SAR) is a technique BRL's CAD Package employs for imaging distant objects. While standard radars only report target backscatter and range information, SAR techniques can resolve distinct scattering regions of an object by correlating multiple radar samples taken from various positions [*Toomay82*]. Thus, by using this sort of predictive analysis, users can learn how models will look to radar.

local graphics display

Designers working in a highly interactive analysis environment spend most of their time using graphics screens to interact with a model geometry--contemplating it, viewing it, changing it. After a significant design change, one or more analyses may be conducted on the new design, with the output images being displayed on a graphics screen as they are computed. Then, interaction with the model can continue.

Several distinct types of graphics support, though, are required to implement this style of interaction. The specification, viewing, and modification of a model geometry can be done under BRL's CAD Package in a highly interactive mode using the system's solid model editor **mged**. A library of *display manager* modules exists to support a wide variety of hardware types both with and without displaylist capabilities, using a single formalized interface.



All images in this article were generated on an IRIS using tools from the BRL CAD Package. Here, positioned clockwise, are: (1) a tank constructed of plate-glass armor (Kennedy); (2) a fanciful blue-chrome molecule (Muuss and Stay); (3) a view of a room in a museum (Muuss); and (4) procedurally generated clutter (Muuss).

BRL's CAD Package also simplifies local graphics display in yet another way. Many analysis programs produce full-color, shaded images. To free applications programs from the specifics of multiple types of hardware, a hardware-independent framebuffer library called **libfb** has been included in the package. By using this library, each application can be written to perform abstract operations on an idealized 24-bit RGB framebuffer. The library allows one application to open an arbitrary number of framebuffers, where each may be of a different type. The format of the *device* string is "*[host:]*/dev/*device_name[#]*" to designate a hardware device; just "filename" will do to designate a file as a virtual framebuffer. If a host name is given, a network connection will be opened to the framebuffer daemon on that machine.

UNIX systems support device-independent plotting in two dimensions with 16-bit signed integer coordinates, using **plot**(5) format plot files. BRL's **libplot3**(3) library provides additional features, including: the ability to use 24-bit RGB colors, points with three coordinates, and routines that take 2D and 3D floating-point values (written in a transportable, machine-independent binary format). What's more, routines for drawing coordinate axes, scaling data, and the like have been added to support traditional data plotting.

networking

Collaboration is an important aspect of most creative endeavors, whether scientific or artistic, and this can often require efficient machine communication. Thus, since the processor architectures of hosts in any given network are likely to be different, careful attention must be given to the format and order in which data is transmitted. But when constructing distributed applications intended to run across a wide variety of machines, it is not possible to predict in advance how the hardware will be coupled. Therefore, either (1) all machines must agree to adopt a single standard for communication, or (2) all machines must be able to convert their data to the internal formats of all other machine types. The first strategy is the one favored by the DARPA Network Research community, so it shouldn't be surprising that the BRL CAD Package continues in this tradition.

Network graphics applications often transmit only integers, either for efficiency or simplified conversion. For scientific computing, though, large dynamic range and many digits of significance are needed, requiring double-precision, floating-point numbers. The most portable representation of a floating-point number would be a printable ASCII string, using at least 23 bytes. The binary form is more compact, requiring a constant eight bytes; this uses one-third the bandwidth of the printable form.

From all of the possible binary formats, the ANSI/IEEE Standard 754 for Binary Floating-Point Arithmetic was chosen for use in the BRL CAD Package. Thus, floating-point values must be converted from the host-specific form to the 64-bit IEEE form before transmission, and upon reception, all floating-point values must be converted from the 64-bit IEEE form to the host-specific form, using library routines. Transmission is in Big-Endian (IBM) byte order. With this standard, it's possible to easily read and write messages in a portable, machine-independent, binary format--which is the principal reason the BRL package has a following in networked environments. A related feature of enormous significance stems from the decision to include the system's 3D floating-point plotting capabilities in **libplot3**, thus providing a machine-independent "plot metafile" capability by which users can "zoom" in by an arbitrary degree to examine extraordinarily fine details.

The BRL Remote Frame Buffer capability, moreover, allows framebuffer operations to be directed to any display screen attached to the network simply by specifying the computer and target display. The details of the network connection are entirely transparent to the user, except for speed variations. The use of framebuffers across fast network connections doesn't appear much different to the user from normal local display performance. This represents a significant convenience to graphics users.

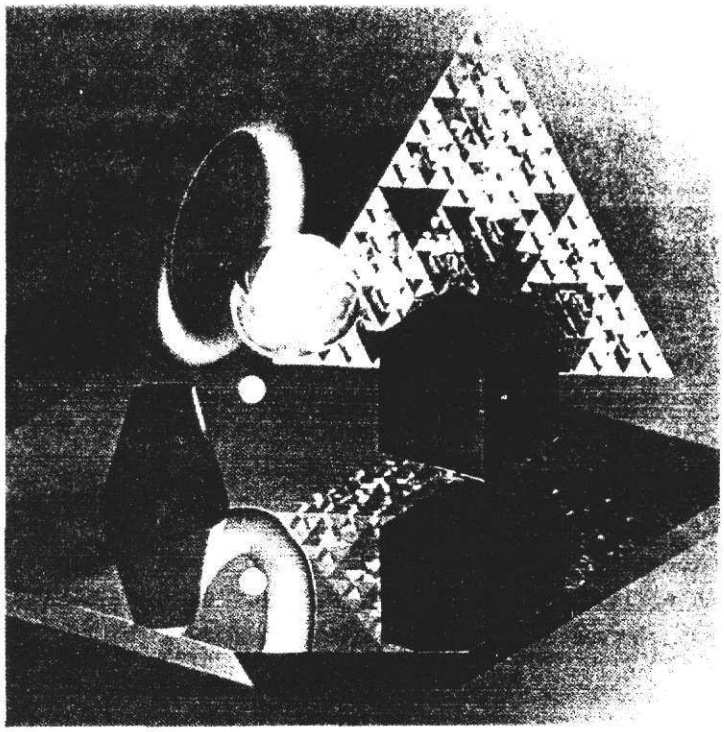
pipeline distributed computation

The careful division of application software into independent tools has resulted in significant rewards for software developers and maintainers alike. For the user, meanwhile, the ability to combine a set of software tools in arbitrary ways might lead to functionality that even the tools' original designers could not have imagined. Traditional operating systems require that each command in a multi-command sequence be run sequentially, with the output from each command being stored in a temporary file. But UNIX allows commands to be connected together using pipes, like so:

```
step1 < input | step2 | step3 > output
```

Not only does this approach avoid temporary files and their storage, it introduces parallelism. The Dual-VAX work done at Purdue University in the early '80s [*Goble81*] popularized multiprocessor UNIX systems. In recent years, such systems have become increasingly common. And they can be conveniently and transparently exploited--without the need to reprogram--by utilizing the UNIX pipeline construct.

A large number of simple tools for manipulating images and framebuffers are provided with BRL's CAD Package. Written in UNIX Software Tools fashion, each of these utili-



The top image here depicts the meeting of the Great Glassner Pyramid and the Moss World (Muuss). Below is a representation of the Moss World in outer space (Dykstra and Butler).

ties performs a single function and is designed to be connected with other tools. Since the tools have a standardized interface, they've proven to be extremely flexible.

The difficulty in learning a large new set of tools is eased in the case of BRL's CAD Package by the use of a common set of flags and naming conventions throughout. The "user interface" is the UNIX shell, together with its conventions for establishing pipes, passing arguments, and so forth. A shell with history recall and screen-oriented command editing is a major convenience.

The effective construction of software tools depends on having standards for the data that's passed between various tools. Binary formats are generally preferred for graphics, and stringent requirements are necessary for file formats so as to keep them portable between different types of machines. The two most common image formats are the eight-bit-per-pixel black-and-white **bw**(5) and 24-bit-per-pixel color **pix** (5) formats. Files have the simplest possible format, with no header, in order to enable the rapid development of new tools. The primary advantage of the headerless format is the convenience it spells when users wish to connect tools using pipes, or to create new tools without resorting to I/O routines specific to the reading and writing of images.

The operations for handling a display device are separated in the package from tools for image handling, which is performed either on pipes or disk files. A common beginning to a pipeline is **fb-pix** to get an image from a framebuffer, with **pix-fb** being used to display the final result. Interaction with framebuffers is supported, allowing changes to be made to colormaps, as well as making it possible to pan and zoom through an image, move a pointer, and add labeling, among other operations. A substantial collection of tools for image manipulation are provided with the package, including statistics routines, histograms, region extractors, rotaters, resizers, and filters. Coupling dataflow tools, moreover, allow one to build complex custom applications without writing any actual code. For example:

```
pixinterp2x -s512  < image.pix | \
pixfilter -s1024 -flo | \
pixmerge - backgnd.pix | \
pixrot -r -i 1024 1024 | \
pix-fb -h
```

says to the system, "take *image.pix* and interpolate

up, then low-pass filter and composite with *backgnd.pix*, then rotate and display on the frame buffer". Notice how no intermediate images are stored in disk files throughout this procedure. This becomes significant when one is required to manipulate 400 MB images, since image processing can take a significant amount of time. There also is another advantage in being able to observe the progress of the computation: as results arrive on the display, the opportunity exists to abort when something goes wrong. In addition to saving computer time, this can save considerable "people time".

general network computing

One of the greatest advantages of the BRL CAD Package is that it allows for efficient operation *across* a network, which can be especially important when that network includes supercomputing resources. Being able to open a window on a Cray from a workstation and obtain the same environment as would appear on other machines in the network is worth a lot. Also, running image processing on a Cray with **rsh**, without having to make special arrangements for moving input files or submitting batch jobs, allows one to harness the power of supercomputing without having to pay a stiff premium in terms of convenience.

The Berkeley **rsh** remote shell command, moreover, has a hidden synergistic power that appears when the command is combined with a collection of good tools. For instance, when logged in on the console of a workstation, this pipeline would produce plot data which would be shown at the local display as:

```
cruncher | pl-sgi
```

In the event that the computation speed of the workstation is not quite sufficiant, a slight variation could be used:

```
rsh Cray.arpa cruncher | pl-sgi
```

By ordering remote execution (via **rsh**) and naming the machine to perform the computation ("Cray.arpa"), the power of an extra machine can be brought to bear. Using **rsh** produces an effect indistinguishable from what would occur were the same operation done locally. Thus, if you were logged in directly to a Cray, and you wished to see an image, you could enter:

```
cruncher | rsh Vax.arpa pl-fb
```

This would send the plot to a framebuffer on "Vax.arpa". To generate a videotape of what happens when a simulation parameter on the Cray is varied, one could use the display and videotape capability on the VAX like so:

```
for parm in `loop 1 100 2`
do
   cruncher $parm | \
   rsh Vax.arpa "pl-fb; vas4 record"
done
```

The **loop** call generates integers between 1 and 100 with an increment of 2. For each value of *parm*, **cruncher** is run, and the plot is sent to the framebuffer on "Vax.arpa", where the video animation controller records the image onto one frame of videotape. Also, consider this variation on the earlier image processing example:

```
pixinterp2x -s512  < image.pix | \
rsh Cray.arpa "pixfilter -s1024 -flo" | \
rsh Alliant.arpa "pixmerge - backgnd.pix | \
rsh Vax.arpa "pixrot -r -i 1024 1024 | pix-fb -h"
```

This interpolates up on the local machine, sending the image to the Cray for filtering, then on to the Alliant for compositing, and finally to the VAX for rotation and display.

With **rsh**, the possibilities of combining tools from different machines are staggering. With the proper infrastructure of computers, operating systems, display hardware, network software, and image processing tools--all connected in compatible ways--the tremendous potential of distributed computing can be easily harnessed, without the need to write any actual code!

the software

The BRL CAD Package is in its third generation, and represents over 150,000 lines of C source code written since 1979. It is highly portable, having run on five distinct generations of hardware using various versions of the UNIX operating system. The software runs on hardware from many different vendors, and is in use at over 350 sites worldwide.

references

[Goble81] G.H. Goble and M.H. Marsh, *A Dual Processor VAX 11/780*, Purdue University Technical Report TR-EE 81-31 (Sept. 1981).
[Muuss87] M.J. Muuss, "Understanding the Preparation and Analysis of Solid Models", *Techniques for Computer Graphics*, ed. D.F. Rogers and R.A. Earnshaw, Springer-Verlag (1987).
[Toomay82] J.C. Toomay, *Radar Principles for the Non-Specialist*, Lifetime Learning Publications, London (1982).

Mike Muuss is the leader of the Advanced Computer Systems Team at the Ballistic Research Laboratory in Aberdeen, MD. Questions or comments pertaining to the BRL CAD Package can be addressed to ***Mike@BRL.MIL.***

MEMORANDUM REPORT BRL-MR-3898

BRL

ISSUES IN AUTOMATIC OBJECT RECOGNITION: LINKING GEOMETRY AND MATERIAL DATA TO PREDICTIVE SIGNATURE CODES

PAUL H. DEITZ
MICHAEL J. MUUSS
EDWIN O. DAVISSON

MARCH 1991

U.S. ARMY LABORATORY COMMAND

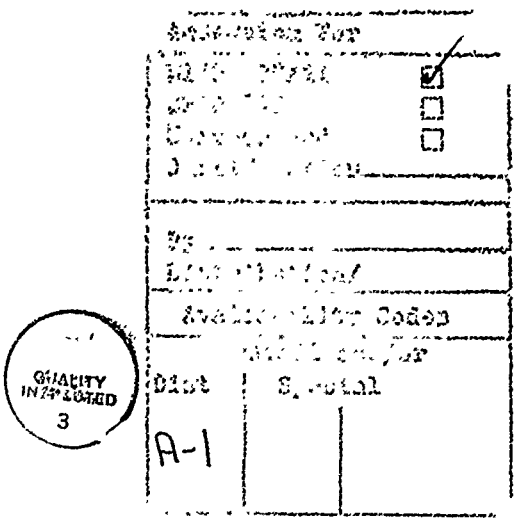
BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

INTENTIONALLY LEFT BLANK.

TABLE OF CONTENTS

INTENTIONALLY LEFT BLANK.

LIST OF ILLUSTRATIONS



INTENTIONALLY LEFT BLANK.

Issues in automatic object recognition: Linking geometry and material data to predictive signature codes‡

Paul H. Deitz
Michael J. Muuss
Edwin O. Davisson

Vulnerability/Lethality Division
US Army Ballistic Research Laboratory
Aberdeen Proving Ground, MD 21005-5066

ABSTRACT

The principal focus of Automatic Object Recognition (AOR) involves the generation of appropriate algorithms to process the output of multi-spectral sensor arrays. Given the high dimensionality that characterizes the signatures of targets of interest, it is normally impossible to satisfy the need for raw signature data by means of measurement records alone. Individual sensor characteristics in conjunction with aspect-angle dependence, target and background configuration (singly and in synergism) and multi-spectral tradeoffs inexorably lead to a requirement for predictive signature modeling methods. By means of this stratagem, a measured signature data base can be leveraged significantly, improving the fidelity of the overall simulation.

Irrespective of the specific representation used for a three-dimensional geometry and material database, rarely does a predictive signature application code read that database directly. Rather, a specific interrogation method is used to pass particular geometric and material attributes to the application code. Clearly the nature of the physics employed in the application is both enabled and constrained by the form of the interrogation process used.

In this paper, several examples of predictive radar codes are given, illustrating several strikingly different ways of linking geometry to applications. Following those examples the interface methods known to the authors will be described. While many of the techniques have already been implemented, some are currently in development. In addition, the utility of various techniques will be related to particular application codes.

1. INTRODUCTION

The principal focus of Automatic Object Recognition (AOR) involves the generation of appropriate algorithms to process the output of what are often multi-spectral sensor arrays. The generation and testing of such algorithms necessarily require the full panoply of desired object (target) signatures as well as undesired and unavoidable (clutter) signatures. Even ignoring the sensor response characteristics (time and space resolution, polarization, noise, etc.), both the target and clutter signature sets are unbounded even for a single frequency, let alone in multi-spectral domains. The unboundedness is strictly true for these signature sets and most probably so even for derived statistical measures.

Given, therefore, the requirements of a typical AOR project for copious signature files for algorithm generation and testing, it is normally impossible to satisfy the need by means of measurement records alone. The single-frequency sensor characteristics noted above in conjunction with aspect angle dependence, target and background configuration (singly and in synergism) and multi-spectral tradeoffs, inexorably lead to a requirement for predictive

‡ Presented at the Advanced Institute Program on Automatic Object Recognition, sponsored by the Society of Photo-Optical Instrumentation Engineers (SPIE), Cocoa Beach, FL, April 21-23, 1990.

modeling methods. By means of this stratagem, a measured signature data base can be leveraged significantly, improving the fidelity of the overall simulation.

For more than twenty years, the **Ballistic Research Laboratory (BRL)** has been utilizing solid geometric modeling methods to support vulnerability, lethality and neutron transport studies of military targets.[1-3] In such item-level studies, target geometry and material information are passed to various application codes to derive certain measures-of-performance. Building on the general paradigm, workers at BRL and elsewhere have extended the general techniques to support many predictive signature models[4-6] including optical, **millimetre wave (MMW)**,† **Infra-Red (IR)**, magnetic and X-ray models.

It is important to note that this type of analysis must generally be supported by a *solid* geometric model. A solid model[7] is a computer description of closed, solid, three-dimensional shapes represented by an analytical framework within which the three-dimensional material can be completely and unambiguously defined. Two major families of solid model representations exist, each with several unique advantages. The first is the **Combinatorial Solid Geometry Representation (CSG-Rep)**.[8] Solid models of this type are expressed as boolean combinations of primitive solids which are geometric entities described by some set of parameters and occupying a fixed volume in space. The second is the **Boundary Representation (B-Rep)**, of which there are two sub-types: (1) *explicit*, where each solid is described by an explicit enumeration of the extent of the surface of the solid; and (2) *implicit*, where the surface of the solid is described by an analytic function such as a Coons patch, Bezier patch, B-spline, etc. Hybrid systems such as the BRL-CAD Package[9] also exist.

Irrespective of the specific representation used for a three-dimensional geometry and material data base, rarely does an application code read that database directly. Rather, a specific interrogation method is invoked to pass particular geometric and material attributes from a source or reference file to the application code. Clearly the nature of the physics employed in the application is both enabled and constrained by the form of the interrogation process used.

In this paper, several examples of predictive radar codes are given, illustrating several strikingly different ways of linking geometry to applications. Following those examples the interface methods known to the authors will be described. Many of the techniques have already been implemented; some are currently in development. In addition, the utility of various techniques will be related to particular application codes.

2. CASE STUDIES IN PREDICTIVE RADAR MODELING

2.1 Examples of Synthetic Aperture Radar Imagery

Early radars were designed to estimate target range and closing rates. For these systems, the main parameter of interest for a given target was **Radar Cross Section (RCS)**. The RCS figure represents the efficiency with which radar waves are scattered back to the receiver. Certain modern radars, when placed on moving platforms such as aircraft, can be used to form two-dimensional images of targets. Radar imagery of this class is called **Synthetic Aperture Radar (SAR)**. A description of an **Armored Fighting Vehicle (AFV)** has been analyzed with a SAR program[10] (to be described below) and the results are shown in Figs. 1 and 2.

Figure 1, left-hand side, shows the AFV as seen by the SAR radar from a (35°, 30°) (Azimuth, Elevation) orientation. A horizontal flight path (left to right) is assumed. The properties of SAR processing are such that following signal detection and manipulation an image is derived which resolves the target in range and cross-range (along the flight path) but not in the remaining orthogonal direction. Thus the final SAR image orientation is similar to the optical rendering shown in the right-hand side of Fig. 1.

---

† Recently, an initiative was made to consolidate Radar Cross Section (RCS) code development sponsored by the Tri-Services and NASA. *The Electromagnetic Code Consortium* has chosen the geometry tools developed by the BRL as the basis for radar codes to be sponsored by this group. See Ref. 6.

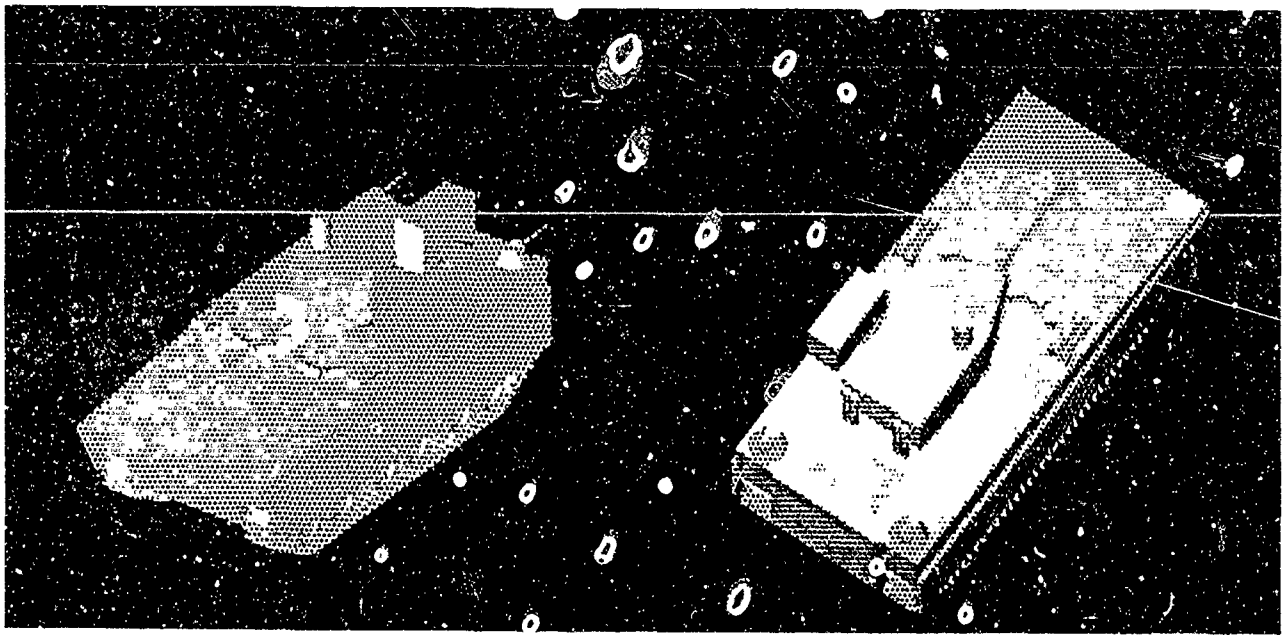


Figure 1. On the left an optical image of an AFV illustrating the radar view of the target (35°, 30°) for a synthetic aperture radar (SAR) simulation. The SAR is modeled as moving in the azimuthal direction (elevation and range constant). On the right, complementary optical image to that shown on the left, the apparent aspect is (215°, 60°) and is suggestive of the SAR reconstructive process when range is plotted against cross-range as in Fig. 2. (From Ref. 10)

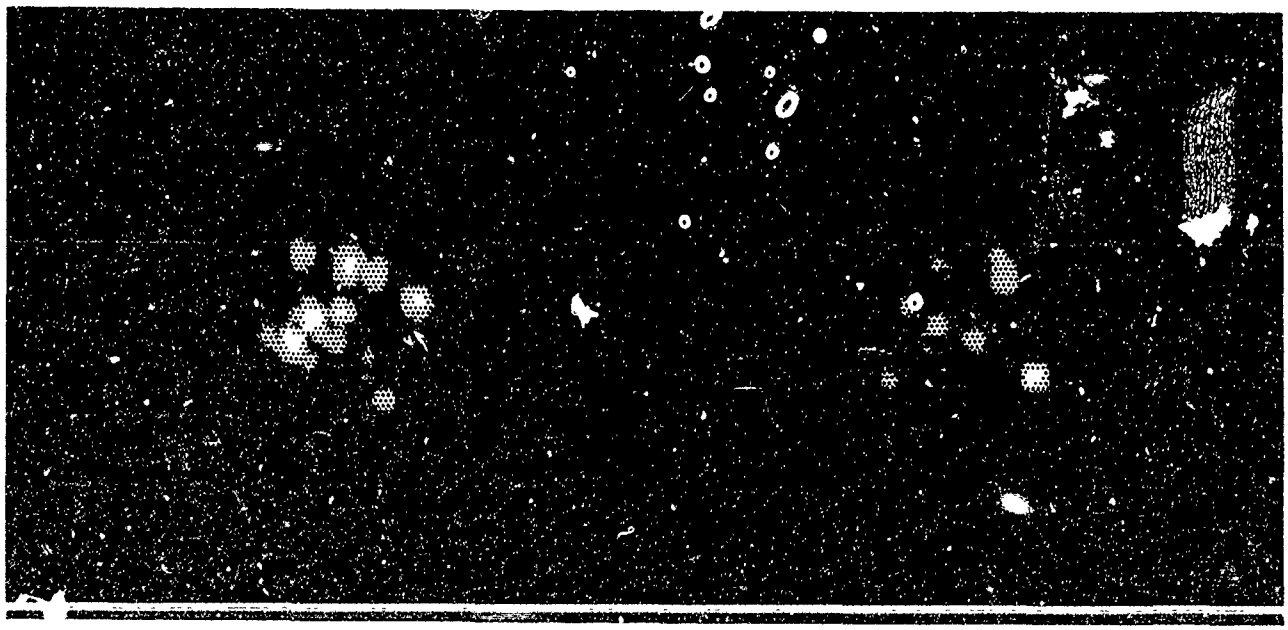


Figure 2. Computed SAR image for target resolution of 10 inches. Left-hand image is for 10 Ghz, Vertical/ Vertical (co-polarization transmit/ receive) modes; right-hand image is for Vertical/ Horizontal (cross-polarization transmit/ receive) modes. (From Ref. 10)

A pair of SAR images for the AFV is shown in Fig. 2. Both images are computed for a target resolution of 10 inches. The left image is the result of a transmit Vertical, receive Vertical polarization mode; the right, for a Vertical/Horizontal polarization mode. In each of these images, the radar signal is propagating from top down. Range information is plotted along the ordinate and cross-range data along the abscissa.

2.2 The SRIM Code of ERIM

Simulated Radar IMagery (**SRIM**) is a high-frequency predictive radar code developed at the Environmental Research Institute of Michigan (**ERIM**).[11] Early versions of the code used a geometry representation scheme for which no convenient editor existed. In 1983, ERIM linked its electromagnetics section of SRIM to BRL's CSG-Rep geometric modeling capability. BRL geometry is described in terms of Boolean combinations (intersections, unions, differences) of primitive objects such as ellipsoids, cylinders, and tori. Ray-tracing is used to extract geometric information from the database. The ERIM-BRL link provides an easy means for generating geometry descriptions through an excellent graphics editor and a natural geometry interrogation technique for determining ray path information. SRIM follows the Geometric Optics (**GO**) paths of rays as they hit and reflect from the target surfaces. For each ray emanating from the radar a piecewise-linear path is traced through the geometry and a history is recorded of hit-point coordinates, normal vectors, principal curvatures, principal curvature directions and a flag designating whether or not line-of-sight exists to the radar. This geometric information is then passed to the electromagnetics portion of the model. At each hit point along the ray path, the incoming illumination wavefront is resolved into parallel and perpendicular components, a complex Physical Optics (**PO**) field contribution at the radar is calculated, taking into account the wavelength of the system and this contribution is then resolved into its vertical and horizontal polarization components. SRIM is thus able to determine the PO field contributions for not only first-surface illumination, but also for contributions due to multiple bounces along the specular directions. If a total radar cross section is desired, this complex return is summed into a total field variable. If a SAR image is desired, the location at which this contribution would appear in an image is calculated and added to the appropriate range/cross-range bin. Image parameters such as resolution, beam weighting and pixel size are specified in a separate file.

If a vehicle on a ground plane is the object of interest, SRIM can generate a clutter model for the ground backscatter and will properly represent the shadowing of the ground plane by the vehicle. In fact, all shadowing effects are correctly represented as a natural result of the ray-tracing paradigm used in this simulation.

2.3 The SARSIM Code of Northrop

SARSIM is another high-frequency synthetic aperture radar simulation code. It was developed at the Northrop Research and Technology Center.[12] In this model, the radar images of the target and surrounding background are computed separately. In a final step the two images are combined *via* processing which reflects appropriate noise, speckle and system-response values. For the geometric calculations an underlying target representation of triangular plates is employed. For a given SAR resolution, each of the triangular plates in the target description is subdivided into panels whose contributions fall into a single SAR resolution bin. For each of these panels in the target description, a PO calculation is carried out to determine the contribution to the corresponding SAR resolution bin. This is also done for a Physical Theory of Diffraction calculation on the panel edges. As of 1987 only the specular (first surface) and diffraction contributions were considered in this model. A simplified shadowing algorithm eliminates any panels partially obscured by other panels. Multiple reflections are not considered yet.

2.4 The TRACK Code of GTRI

Georgia Tech Research Institute (**GTRI**) has developed a radar prediction tool called TRACK.[13] The geometry used by TRACK is based on a subset of objects supported by the GTRI MAX geometry editor. A hybrid representation of simple shapes such as ellipsoids, plates, frusta and special radar-specific constructs such as dihedrals and trihedrals are accepted by the TRACK code. TRACK has a series of subroutines referred to as CROSS which calculate the field contribution of the individual scatterers in a MAX geometry file. CROSS uses PO to calculate the fields for polygonal flat plates, right circular frusta, general ellipsoids, ogives and toroids. A combination of Geometric and Physical Optics is use to calculate the fields for dihedral and trihedral shapes. CROSS also predicts diffraction from straight and curved edges by methods of equivalent currents to avoid problems associated with caustics. The returns from the individual scatterers can be summed coherently,

noncoherently, or retained in complex form for postprocessing. Shadowing is handled by ignoring all plates that face away from the radar and by excluding from the electromagnetics calculations any scatterer that is completely obscured by other geometric elements. When an element is partially covered, the full energy of that element is returned, which can sometimes be a source of error. For post-processing, a file with scatterer types, location, size and orientation is generated. This can then be used in numerous scenarios, such as SAR, doppler radar, and **ISAR** (Inverse **SAR**).

2.5 Discussion of Approaches

Each of the aforementioned high-frequency scattering codes required compromise. The choice of geometric representation and interrogation method led to advantages and shortcomings. Choosing a particular geometric approach often requires approximations in the implementation of the physics of an analysis. Likewise, a particular formulation of a physical analysis can force the acceptance of a simplified and less general geometry representation for the sake of getting some portion of the problem solved.

SRIM uses the information gathered by ray-tracing to determine a scattered field. Fields are calculated locally so that the ray-trace sampling density can be chosen small enough to guarantee that each contribution to the scattered field will lie in a unique resolution cell, a feature not guaranteed by the GTRI model. While the ray-tracing technique is computationally intensive, the automatic shadowing and multiple bounce calculations are attractive features. If two modeled surfaces are at a right angle to each other, the SRIM code will calculate a scattered field that is appropriate for a dihedral without requiring the explicit designation of a dihedral element in the geometry file. This approach does have the severe cost of requiring ray-tracing densities which are frequency dependent, so that the number of rays which must be cast at the target description for a 94 GHz radar calculation is on the order of 100 times the number of rays needed at 10 GHz. Originally the code simply point-sampled the field contribution at a location, but this approach suffered from aliasing problems due to under sampling high spatial frequency (fine detail) geometric elements. Improvements have been made so that the field contribution is integrated over a planar approximation at the scattering surface, so that field calculations are less sensitive to ray-trace sampling densities. Even so, with a ray-tracing approach, one can still run the risk of not adequately sampling the fine detail in the geometric model, although increasing sampling densities with ray-tracing makes the inter-ray sampling distances arbitrarily small. It is possible to place an upper bound on the size of geometric objects that will be under sampled; with knowledge about the distribution of feature sizes in the geometry, the ray spacing can be chosen so as to make it statistically unlikely to miss any details.

Only PO fields are calculated for SRIM. Diffraction effects occurring at edges are not calculated because the ray-tracing interrogation method does not provide the required edge information. Some of the representational schemes discussed later will show how this might be accomplished with an alternate form for extracting information from the model geometry.

One advantage of the SARSIM approach to field calculations is that integration takes place over each facet or panel in a target description and all of the modeled geometry is guaranteed to be included in the calculation. SRIM can not guarantee this. Likewise, if one has adopted a particular resolution cell size and subdivided the triangular plates into panels appropriate for the resolution cell size, there will be no increase in the number of panels for an increase in the frequency of the radar. On the other hand, in modeling the target with flat plates, one must approximate any curved surfaces and thereby introduce some uncertainty about the fidelity of the geometry itself, since flat plate representations of curved surfaces introduce edges where no real edges exist. Thus, each edge must be marked by the geometry modeler, to indicate if the edge is an actual edge, or a modeling artifact. For radar diffraction calculations this criterion must be used to trigger the inclusion or exclusion of edges.

Multiple reflections are not modeled in SARSIM, so any radar signature dominated by multiple bounces of energy within the target will be poorly modeled. Many ground vehicles, especially tracked vehicles, exhibit dramatic dihedral and trihedral returns, making multiple-bounce considerations a primary concern. The use of piecewise-linear target representations does lend itself to polygonal projection for determining multiple path interaction, thereby avoiding ray-tracing. However, such an approach introduces its own set of difficulties. These techniques require significant computation to find all the facet-to-facet "form factors". For a geometry with N facets, a solution to the global illumination problem requires a system of N equations in N unknowns to be solved.

With most targets of interest the value of N is quite large, and performing Gaussian elimination on the N-by-N matrix is often prohibitive.

The GTRI radar simulation provides highly accurate electromagnetic calculations for most of its primitive geometry types. Of the three models considered here, it provides the most detailed calculations for certain simple objects. For many modeling scenarios this approach gives excellent results. However, the technique for combining these component field calculations for a complex geometric object fails to address some important issues. The basic approach of modeling complex geometry with simple objects for which highly detailed electromagnetic field calculation are known, has been used for many years.[14] For simple shapes, *e.g.* rockets, or for objects where a few scatterers with wide spatial separation dominate the returns, this has been a highly successful technique. When the target under consideration has non-convex complexity such as seen in a tank, the effects of multiple-reflection scattering cannot be ignored. Although the GTRI model has dihedral and trihedral geometric primitives to account for some multiple reflection scattering, the modeler must provide the information of how large and where these objects should be. Even with this feature, multiple reflection paths may escape the modeler's attention or be of such complexity as to be insufficiently modeled by dihedrals or trihedrals. To deal with this type of scattering with a general-purpose code that does not require operator intervention, it would be necessary to appeal to another geometric model of higher resolution to find the multiple reflection paths.

One other shortcoming of this method is the coarseness of the criterion for determining when one primitive obscures or shadows another. The GTRI method does not sufficiently subdivide the scattering geometry, so that full-power returns to the radar are considered to exist even when the scattering object is almost completely obscured. Only when a primitive is totally obscured is the electromagnetic scattering omitted from the field calculation. Notice that even if it were geometrically simple to determine what portion of a given solid is obscured, this electromagnetics calculation method would not provide the field contribution from the partially visible portion. Still worse problems arise for imaging radar simulations if the scattering object is geometrically larger than a resolution cell, since a given scatterer has only one field value and that field value cannot be distributed over more than one cell. This all-or-nothing field calculation method is a consequence of the physics being constrained by the chosen geometry interface.

A geometric representation method and an interface of that method to the scattering calculations have been selected in each of the three codes just discussed. These choices clearly have advantages and liabilities. Any code requiring geometric information will likewise be limited in some areas and enhanced in others by the choices of the geometric representation and the interface of that geometry to the analysis. Each analysis code described so far employs only one of these geometry/analysis code links, but significant benefits might be gained from analysis codes that use two or more of these interfaces simultaneously. Examples of the information that can be readily provided from various geometry representations and interfaces will be discussed in the following sections.

3. GEOMETRIC INTERFACES

In what follows a number of methods are described by which geometric and material data are supplied to applications codes. Where possible, the interface methods developed and supported by the BRL-CAD geometry tools[9,15] will be used to describe and illustrate these processes. A goal of BRL-CAD has been to provide a general and open analysis environment in which a variety of interrogation interfaces are supported so that diverse application codes can be driven from a single, unified geometric model.[16] The **APPENDIX** gives a brief overview of the BRL-CAD modeling environment and some strategies which have guided its development.

3.1 Ray-tracing

Rays begin at a point $\vec{P}$, and proceed infinitely in a given direction given by the unit normal vector $\vec{D}$. The *direction vector* or *direction cosines* for the ray (D_x, D_y, D_z) are the cosines of the angle between the ray and each of the Cartesian axes. Any point $\vec{A}$ on a ray may be expressed as a linear combination of $\vec{P}$ and $\vec{D}$ by the formula

$$\vec{A} = \vec{P} + k * \vec{D}$$

where valid values for k are in the range $[0, \infty]$.

The traditional approach to ray-tracing has been batch-oriented, with the user defining a set of "viewing angles", initiating a large batch job to compute all the ray intersections and then post-processing all the ray data into some meaningful form. However, the major drawback of this approach is that the application has no immediate control over ray paths, making another batch run necessary for each level of reflection, etc.

In order to be successful, applications need: (1) interactive control of ray paths, to naturally implement reflection, refraction and fragmenting into multiple subsidiary rays and (2) the ability to fire rays in arbitrary directions from arbitrary points. Nearly all non-batch implementations have closely coupled a specific application (typically a model of illumination) with the ray-tracing code, allowing efficient and effective control of the ray paths. The most flexible approach of all is to provide the ray-tracing capability through a general-purpose library and make the functionality available as needed to any application. For example, the decision of when a ray should be reflected, transmitted, or absorbed should be entirely under the control of the application program. A set of sample ray histories that might result from an application exerting such interactive control is shown in Fig. 3.

3.1.1 RT Library Interface: The third generation ray-tracing capability in the BRL-CAD Package is a set of library routines in **librt** to allow application programs to intersect rays with model geometry. There are two parts to the interface: "preparation" routines and the actual ray-tracing routine. **rt_dirbuild()** opens the database file and builds the in-core database table of contents. **rt_gettree()** adds a database sub-tree to the active model space, and can be called multiple times to join different parts of the database together.

To compute the intersection of a ray with the geometry in the active model space, the application must call **rt_shootray()** once for each ray. Ray behaviors such as perspective, reflection, refraction, etc, are entirely determined by the applications program logic, and not by the ray-tracing library. The ray-path specification determined by the applications program is passed as a parameter to **rt_shootray()** in the **application** structure, which contains five major elements: the vector a_ray.r_pt ($\vec{P}$) which is the starting point of the ray, the vector a_ray.r_dir ($\vec{D}$) which is the unit-length direction vector, the pointer *a_hit() to an application-provided routine to be called when some geometry is hit by the ray, the pointer *a_miss() to an application-provided routine to be called when the ray does not hit any geometry, and the variable a_onehit. In addition, there are various locations for applications to store state information such as recursion level, intermediate color values, and cumulative ray distance.

When the a_onehit variable is set to zero, the ray is traced through the entire model. Applications such as lighting models may often only be interested in the first object hit; in this case, a_onehit may be set to the value one to stop ray-tracing as soon as the ray has intersected at least one piece of geometry. Similarly, if only the first three hits are required (such as in the routine that refracts light through glass), then a_onehit may be given the value of three. Then, at most three hit points will be returned, an in-hit, an out-hit and a subsequent in-hit. When only a limited number of intersections are required, the use of this flag can provide a significant savings in run-time.

The **rt_shootray()** function is designed for full recursion so that the application provided a_hit()/a_miss() routines can themselves fire additional rays by recursively calling **rt_shootray()** before deciding their own return value. In addition, the function **rt_shootray()** is fully capable of operating in parallel with other instances of itself in the same address space, allowing the application to take advantage of parallel hardware capabilities where such exist.

3.1.2 Sample RT Application: A simple application program that fires one ray at a model and prints the result is included below, to demonstrate the simplicity of the interface to **librt**.

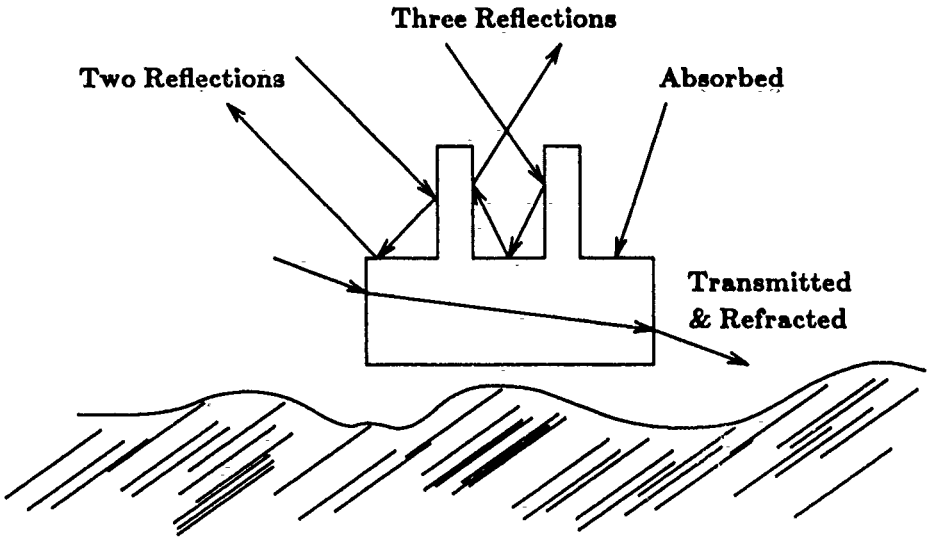


Figure 3. Application Control of Multiple Bounces

```
struct application ap;
main() {
    rt_dirbuild("model.g");
    rt_gettree("car");
    rt_prep();
    VSET( ap.a_point, 100, 0, 0 );
    VSET( ap.a_dir, -1, 0, 0 );
    ap.a_hit = &hit_geom;
    ap.a_miss = &miss_geom;
    rt_shootray( &ap );
}
hit_geom(app, part)
struct application *app;
struct partition *part;
{
    printf("Hit %s", part->pt_forw->pt_regionp->reg_name);
}
miss_geom(){
    printf("Missed");
}
```

3.1.3 Ray Intersection Data: If a given ray hits some model geometry, the a_hit() routine is called and is provided a pointer to the head of a doubly-linked list of **partition** structures. Each **partition** structure contains information about a line segment along the ray; the partition has both an "in" (**pt_inhit**) and an "out" (**pt_outhit**) hit point. Each hit point is characterized by the hit distance **hit_dist**, which is the distance k from the starting point **r_pt** along the ray to the hit point. The linked list of **partition** structures is sorted by ascending values of **hit_dist**. As a result of this definition, the "line-of-sight" distance between any two hit points can be determined simply by subtracting the two **hit_dist** values. This will give the distance between the hit points, in millimeters.

If the variable a_onehit was set non-zero, then only the first a_onehit hit points along the partition list are guaranteed to be correct; any additional hit points provided should be ignored. This is usually important only

when a_onehit was set to an odd number; in this case the value of pt_outhit in the last **partition** structure may not be accurate and should be ignored.

If the actual 3-space coordinates of the hit point are required, they can be computed into the **hit_point** element with the C-language version of $\vec{A} = \vec{P} + k * \vec{D}$:

VJOIN1(hitp->hit_point, rayp->r_pt, hitp->hit_dist, rayp->r_dir);

3.1.4 Surface Normals: As an efficiency measure only the hit distances are computed when a ray is intersected with the model geometry. For any hit point, the surface normal at that point can be easily acquired by executing the C macro:

RT_HIT_NORM(hitp, stp, rayp);

In addition to providing the unit-length outward-pointing surface normal in struct **hit** element **hit_normal**, this macro also computes the 3-space coordinates of the hit point in struct **hit** element **hit_point**.

3.1.5 Gaussian Curvature: For any hit point, after the surface normal has been computed, the Gaussian surface curvature at that hit point can be acquired by executing the C macro:

RT_CURVE(curvp, hitp, stp);

At the hit point, there exists exactly one pair of orthogonal directions also orthogonal to the surface normal $\vec{N}$ for which the values of c take on the minimum and maximum values c_1 and c_2. c_1 and c_2 are the inverse radii of curvature and $|c_1| \leq |c_2|$, *i.e.* c_1 is the most nearly flat principle curvature. A positive curvature indicates that the surface bends toward the (outward pointing) normal vector $\vec{N}$ at the hit point. A **curvature** structure has three elements, the unit vector **crv_pdir** (or $\vec{A}$) pointing in the direction of the first principle curvature, the scalar **crv_c1** (or c_1) giving the curvature in the first principle direction and the scalar **crv_c2** (or c_2) giving the curvature in the second principle direction $\vec{B}$. The second principle direction $\vec{B}$ is implied and can be found by taking the cross product of the normal with **crv_pdir**, *i.e.*, $\vec{B} = \vec{N} \times \vec{A}$.

3.1.6 U-V Mapping: Each primitive solid can be considered to be bounded by one or more *regular surfaces*. Each regular surface is defined as the locus of points $\vec{S}(u,v)$ depending on two real parameters u and v which range from 0.0 to 1.0 inclusively. These parameters form the coordinates of a two-dimensional Cartesian u,v-plane. A given (u,v) coordinate will appear only once on each regular surface, but in objects with more than one surface that same (u,v) pair may appear at more than one place on the object. The (u,v) coordinate of the hit point is returned in **uvcoord** structure elements uv_u and uv_v. For any hit point, after the value of **hit_point** has been computed, the (u,v) coordinates of that point can be acquired by executing the C macro:

RT_HIT_UVCOORD(ap, stp, hitp, uvp);

For some simple optical rendering applications , it is sometimes desirable to create a mapping between the coordinate system on the surface of an object to coordinates on a plane. This is generally used to drive simple, two dimensional *texture mapping* algorithms. The most common application is to extract a "paint" color from a rectangular image file at coordinates (u,v), and apply this color to the surface of an object. These parameters can also be used to simulate the effect of minor surface roughness using the *bump mapping* technique. Here, the u and v coordinates index into a rectangular file of perturbation angles; the surface normal returned by RT_HIT_NORM() is then modified by up to ±90 degrees each in both the u and v directions, according to the stored perturbation.

In addition, the approximate "beam coverage" of the ray in terms of the parameters (u,v) is returned in the structure elements uv_du and uv_dv. These approximate values are based upon the ray's initial beam radius (a_rbeam) and beam divergence per millimeter (a_diverge) as specified in the application structure. These delta-u and delta-v values can be helpful for anti-aliasing or filtering areas of the original texture map to produce an "area sample" value for the hit point.

3.2 Topological Representation

Some predictive radar signature codes, such as the TRACK code of GTRI discussed above, do not operate directly on a solid geometric representation of an object. Instead, they rely on the fact that large radar returns occur primarily due to the existence of dihedral and trihedral structures in the object. Rather than describing a vehicle simply as a collection of these topological structures, one can analyze a three dimensional solid model to locate all instances of the topological features of interest. For example, the software could locate planar face elements, edges where two locally planar elements join to make a dihedral, edges where three locally planar elements join to make a trihedral, etc. Then this list of topological features becomes the input to the feature-based analysis code.

Due to the rather broad set of possible interpretations of the term "feature", each kind of topological feature extraction is itself considered an application program and, therefore, is not a standard part of the interrogation library. The process of topological feature extraction is currently programmed using the ray-tracing interrogation features described above.†

3.3 3-D Surface Mesh

Combinatorial Solid Geometry (CSG) models are formed by the boolean combination of "primitive" solids.[7] For example, a plate with a hole is most easily modeled as a plate primitive minus a cylinder primitive. It is important that in CSG models, there is no explicit representation of the surfaces of the solids stored; indeed, for complex boolean combinations of complex primitives, some of the resultant shapes may have very convoluted topology and surfaces that may be at best high degree polynomials.

There are many applications that would benefit from being able to express an *approximation* of these complex shapes created using CSG modeling as a collection of planar N-sided polygons (N-gons) which together enclose roughly the same volume of space as the original CSG solid. The most obvious such application is to drive polygon-based rendering routines (lighting modules) for predictive optical signatures. On many modern workstations there is direct hardware or firmware support for high-speed rendering of polygons. In addition, there are whole collections of polygon-based predictive infrared and radar signature programs. Some of the most accurate radar signatures have been calculated using the Method of Moments.[17] This approach requires a three dimensional surface tessellation to sub-wavelength resolution over an entire geometric model.

A sensible strategy for converting a CSG model to the equivalent approximate three dimensional surface mesh is to tackle the problem in two parts. First, a routine has to be written to convert each of the primitive solids into tessellated form. Second, a routine has to be written to take two tessellated objects and combine them according to a boolean operation (union, intersection or subtraction) back into a consistent set of solid tessellated objects. Until very recently, this second step has proven extremely difficult. The topology of solid tessellated objects has traditionally been represented using the "winged-edge" data structure. Within the winged-edge representation, an edge represents the boundary or intersection between exactly two faces. Unfortunately, this structure fails to handle other valid configurations, such as an edge being shared by four faces, or an edge being part of only a single ("dangling") face. These other configurations arise when the topology of an object is not that of a simple 3-manifold, *i.e.* when the topology of the object can not be mapped to a sphere. These non-3-manifold conditions arise in the construction of finite element meshes, and from the use of the boolean intersection operation.

The inability of the winged-edge data structure to represent non-3-manifold configurations resulted in the development of a more general data structure[18,19]. This new data structure has been dubbed alternately the "radial-edge", "Non-Manifold Geometry" (NMG), or "*n*-Manifold Geometry" data structure. The radial-edge representation provides topological links between *all* faces which share an edge. This single representation has the ability to handle *n*-manifolds ($\mathbf{M}^n$) for $0 \leq n \leq 3$: 3-manifolds (solid objects), 2-manifolds (lone faces, not part of a solid), 1-manifolds (lone edges, not part of a face) and 0-manifolds (lone points, not part of an edge).

† See Ref. 5, Figs. 8 and 9.

Consider the intersection of two solids that share only a single face, edge, or vertex. The result of the intersection will be a manifold object of dimension less than three. The winged-edge data structure is unable to represent these boolean results, because it requires every edge to adjoin exactly two faces. Thus, the winged-edge representation is not closed under boolean operations. To overcome this lack of closure, winged-edge systems substitute *regularized boolean operators* which are defined to produce only 3-manifold results; all lesser dimension results are discarded. In contrast, because all manifold objects of dimension less than three can be represented using the NMG data structures, NMG objects are closed under boolean operations.

Employing the NMG representation for faceted solid objects gives rise to the rich set of possibilities diagramed in Fig. 4. From this diagram it should be clear that the final evaluated NMG solid object can be employed in a variety of ways. The primary use will be for input to analysis codes that need an approximate 3-D surface mesh of the solid model. In this case, the NMG objects are sent across the interface, either directly into an application, or *via* a triangulator that turns the planar N-gon faces of the NMG objects into simple triangle lists and thence to applications such as SARSIM. However, a very powerful second use will be to create new faceted shapes which are then stored back in the database as new geometric objects, suitable for future editing or analysis.

While a detailed description of the NMG data structures is beyond the scope of this paper,[20] there are several advantageous properties of the NMGs that are worth mentioning. The NMG representation maintains full topology information, so that the relationships between vertices, edges, loops, faces and shells are continuously available. The geometry information associated with a planar face is the plane equation (which includes the outward-pointing surface normal); the plane equation does not have to be re-derived from the vertices. For applications that would prefer visual realism rather than geometric fidelity, there is room in the vertex geometry structure to carry around a "phony" normal for each vertex, suitable for intensity interpolation shading algorithms used in Gouraud shading,[21] or for normal-vector interpolation shading algorithms.

One of the most exciting current research projects at BRL is the extension of the NMG framework to permit faces either to be planar N-gons, or trimmed **N**on-**U**niform **R**ational **B**-**S**plines ("trimmed **NURBS**"). This will permit many of the tessellation operations to be implemented exactly, rather than as approximations. This will also permit solids to enjoy the economy of having most faces be represented as planar N-gons, which are very compact and efficient to process, while those few faces that require sculptured surface shape control can be represented as trimmed NURBS. This combination provides both efficiency and full shape control in the rich n-manifold topological framework; a combination that does not exist in any current commercial CAD system.

3.4 3-D Volume Mesh

Many forms of energy flow analysis, such as heat flow, vibrational analysis (acoustic energy flow), and stress analysis require the use of 3-D **F**inite-**E**lement **M**esh (**FEM**) techniques. While there has been some work on using the ray-tracing paradigm to construct finite element and finite difference meshes[22] it has been difficult to deal with high spatial frequency (fine detail) portions of the model. In particular, meshing small diameter pipes is problematic; undersampling can cause the pipe to incorrectly be separated into multiple pieces.

In order to improve on the current state of affairs, it seems necessary to provide support for the generation of volume meshes directly as part of the application interface. This would provide the meshing algorithm to have unrestricted access to the underlying geometry, the space partitioning tree, and other internal data in order to perform a better job.

Even more promising still would be a strategy that takes advantage of the NMG support. A first pass might tessellate the model and evaluate the booleans to produce a surface mesh. The second pass would then take the surface mesh and fill the interior (or exterior) volumes with appropriately chosen volume elements. A very good fit could probably be achieved using only parallelepiped ("brick") elements and 20-node "superelements". The brick elements would be used to fill interior volume that does not border on a face, and the superelements would be used for volume that contacts a face. Recourse could be made back to the underlying geometry (perhaps *via* firing a few well chosen rays) to get the curvature of the superelement faces to match the curvature of the underlying primitive, rather than having to rely strictly on the NMG planar-face approximation.

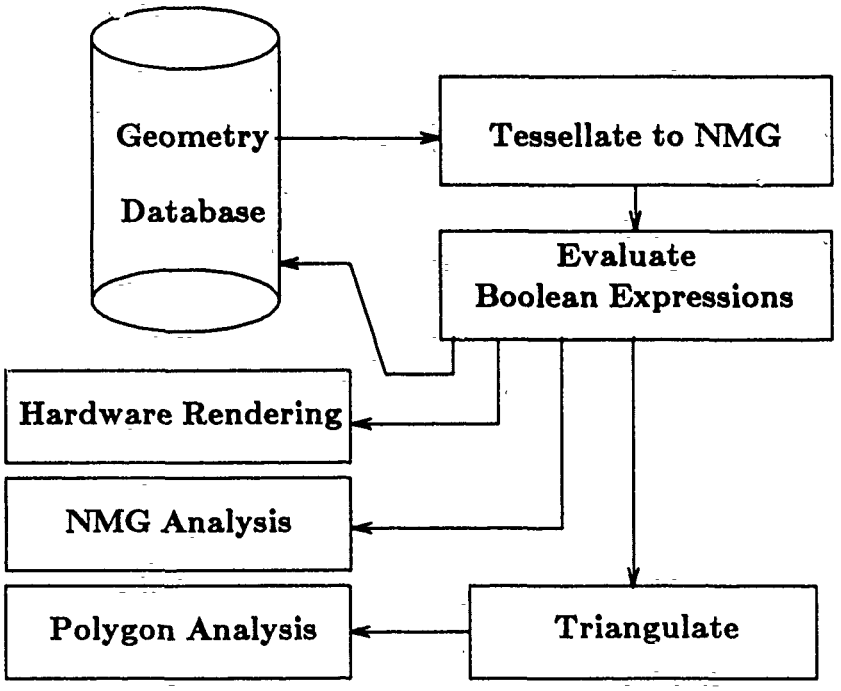


Figure 4. Logic flow for n-Manifold Geometry (NMG) processing.

3.5 3-D Volume Elements (Voxels)

A representation which is similar to the finite-element mesh is based on Volume Elements (**VOXELS**). There are two distinct kinds of voxels. The first kind of voxels can be considered a special case of volume meshing discussed previously, in which the model is "diced" into a large collection of homogeneous parallelepiped ("brick") elements. As one example, ERIM has a utility program which uses ray-tracing to convert BRL CSG-format geometry to this kind of voxel representation to feed a first-principles IR model.[23] In contrast to the ERIM's voxel modeling approach, the Physically Reasonable Infra-Red Signature Model (PRISM) predictive IR code, developed at the Keweenaw Research Center, is based on a geometry of flat plates.[24] Two codes linking the BRL CSG-format geometry to PRISM have been developed: the TACOM FRED editor[25] and the BRL **irprep** program.[26]

A distinctly different form of voxel representation is based upon the use of 8-way binary space subdivision stored using an "oct-tree" data structure. In this technique, the model is enclosed in a bounding box. The bounding box is evenly split along the X, Y and Z axes to form eight smaller boxes. This algorithm is applied recursively so that all boxes which are neither entirely full nor entirely empty are repeatedly split, until the size of the voxels satisfies some termination condition. In this way, small voxels that lie along the surface of objects can fit arbitrarily tightly to the surface, while the interior of an isomorphic region will be contained primarily in a single large voxel.

The oct-tree representation provides the application program with a homogeneous geometric representation based entirely on cubes of varying size. Having such a homogeneous representation can often greatly ease the task of algorithm development. On the other hand, achieving a good approximation of curved objects using cubes requires a huge number of voxels to be used, resulting in very large voxel datasets, and an exponential increase (order N^6) in the number of element-to-element equations to be solved. The oct-tree approach to IR signature generation is employed in the BRL-CAD program **lgt**.[27]

3.6 Homogeneous Trimmed B-Splines

When support for trimmed NURB faces has been added to the NMG capability, it will be possible to represent all existing primitives either with exact rational B-spline versions, or with very good rational B-spline approximations. This could be done even for faces that were completely planar.

This offers the hope that it might be possible (albeit memory intensive) to convert an entire CSG solid model into a homogeneous collection of non-uniform rational B-spline faces organized in a n-manifold topological data structure. In addition to the conceptual simplicity afforded by having a uniform representation for shape, this affords the opportunity to create new analysis codes that can process curved surfaces, yet at least initially only have to deal with one kind of shape. This would also provide a very direct and natural interface to spline-based and Bezier-patch[28] based modeling systems.

3.7 Analytic Analysis

Given a homogeneous geometric representation such as the Trimmed B-Splines just discussed which also has an analytic representation, a further processing capability arises. Rather than interrogating the data base by means of sampling or subdivision techniques, the direct mathematical manipulation of the source geometry through its parametric representation becomes possible. Calculations of physical properties requiring integration over a surface can often be evaluated with greater accuracy using an explicit analytic calculation than could be provided by numerical methods. While this may be difficult in general due to the complexity of a parametric expression, some classes of surface representations may be good candidates for this approach. Splines, for example, are piecewise-polynomial functions which have relatively simple Fourier transform representations. Since 2-D spatial Fourier transforms arise frequently in far-field electromagnetic scattering calculations, exploitation of the parametric spline form is of interest in predictive scattering calculations. Direct use of spline parameters in a PO scattering model is part of the methodology used at the Aircraft Division of The Northrop Corporation.[29]

Support for this tack may be inferred from the work of Schneider and Peden who have exploited analytic methods for calculation of radar cross sections using dielectric ellipsoids[30] in the detection of buried targets.[31] Here the approach involves the approximation of the geometry of interest with a shape for which the analytic solution of the electromagnetic scattering problem can be solved. Great care must be taken to insure that the errors introduced by the geometry are a small perturbation of the desired solution.

With the rapidly developing pote :al of symbolic calculation, treatment of seemingly impossible formulas resulting from the geometry/physics ir'eraction may become tenable. This could help to reduce the trend towards employing numerical methods at the onset of a problem and avoid some of the accompanying instabilities and errors.

4. SUMMARY

In this paper we have discussed some issues basic to the prediction of signatures in support of Automatic Object Recognition, in particular the way in which three-dimensional geometry and material data are linked to certain applications. This point was illustrated *via* a discussion of three predictive radar codes, each designed for essentially the same application, but each nevertheless employing unique geometric methods. The descriptive approach and the manner in which each is linked to the physics of the codes has been shown to both enable and constrain algorithmic exploitation.

The methods for linking geometry to applications codes were described in turn. The BRL-CAD package was used to illustrate each approach. As described in the **APPENDIX**, the strategy utilized in BRL-CAD is based on an inhomogeneous collection of closed-surface geometries, of which the variety of shapes is constantly expanding. By this tack, in addition to using geometry files created with BRL-CAD, three-dimensional geometry files that were developed under other systems can either be mapped into the BRL-CAD data base ***exactly*** (if the corresponding geometric forms already exist) or the data base can be extended to support any important new forms (so that the conversion becomes an exact mapping as well). By this method the enormous costs of geometry generation can be

recovered; any model developed on a true 3-D modeling system can be transferred through file importation to the general and open environment of BRL-CAD.‡

Another important point that was emphasized was the possible utility of each particular interface. A related goal, therefore, of BRL-CAD has been to support *all* such interfaces from the extensible data base of geometry types. Many of the forms have been implemented; the NMGs which will provide homogeneous polygon support are in the final stages of development.

By these developments it is our intent to support the AOR community to the maximum extent possible with the tools needed to develop the multi-spectral, multi-sensor simulations critical to autonomous-sensor applications.

APPENDIX

The solid modeling package called BRL-CAD[9,15] was originally developed to support the input requirements of vulnerability and lethality modeling at the Ballistic Research Laboratory. Its capabilities are briefly summarized here:

- BRL-CAD is composed of more than 200,000 lines of portable C language source code which support:
  - Solid geometric editor (mged)
  - Ray tracing utilities
  - Lighting models for a variety of frequencies
  - Many image-handling, data-comparison, image-processing and other supporting utilities
- The set of closed-surface (inhomogeneous) geometrical representations supported by BRL-CAD include:
  - The original Constructive Solid Geometry (CSG) BRL data base
  - Non-Uniform Rational B-Spline Surfaces (NURBs)
  - The faceted data representation (PATCH) developed by Falcon/ Denver Research Institute and used by the Navy and Air Force for vulnerability calculations.
- It supports association of material (and other attribute properties) with geometry which is critical to subsequent applications codes.
- It supports a set of extensible interfaces by means of which geometry (and attribute data) are passed to applications:
  - Ray casting
  - Topological representation
  - 3-D Surface Mesh Generation
  - 3-D Volume Mesh Generation
  - Analytic (Homogeneous Spline) representation
- Source code for BRL-CAD has been distributed to more than 650 computer sites world wide throughout Government, Industry and Academia.
- In addition to the vulnerability and signature codes generated by the BRL, many BRL-CAD based applications codes have been built by others, including applications developed by workers at TACOM/ Keweenaw Research Center, ERIM, Northrop, MITRE, University of Illinois and scores of other sites.

Figure A1 gives a general layout of the BRL-CAD database representation. A number of fundamental strategies are key here. First, the primary data base is inhomogeneous; that is, many types of geometric representations are allowed so long as they represent fully enclosed space. New shapes are added to the geometric

‡ An important caveat here is that the mathematical form of the geometric representation and data base to be imported must be known. Unfortunately, most commercial CAD vendors are unwilling to provide this information for files generated by their systems, since that might free the user from "vendor lockin".

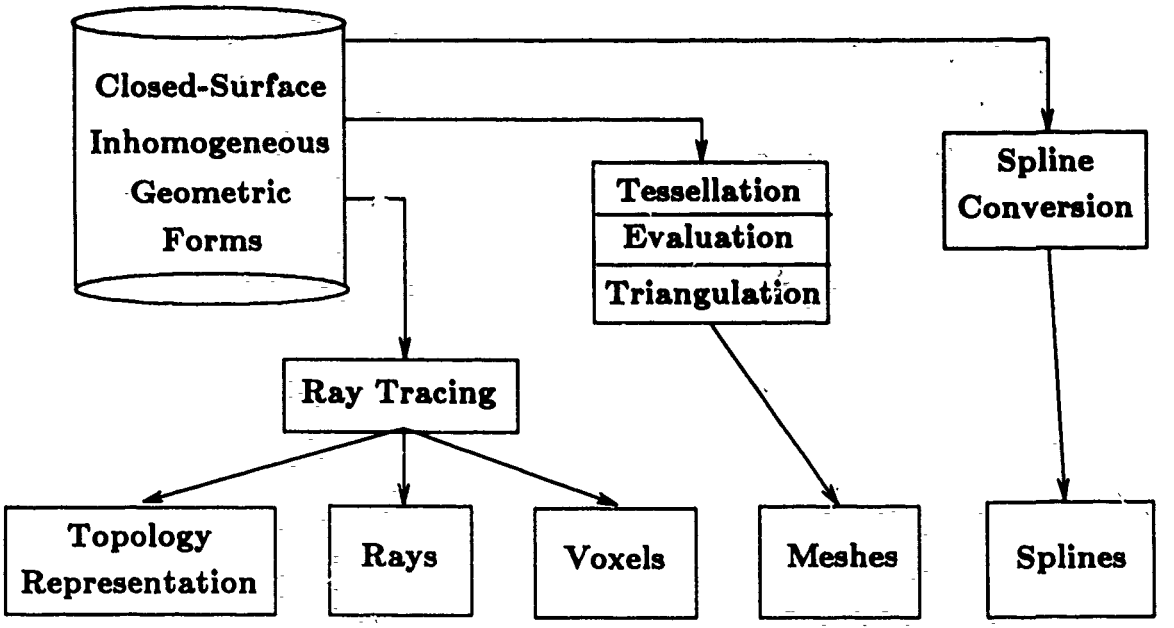


Figure A1. Organization of BRL-CAD Database Representation.

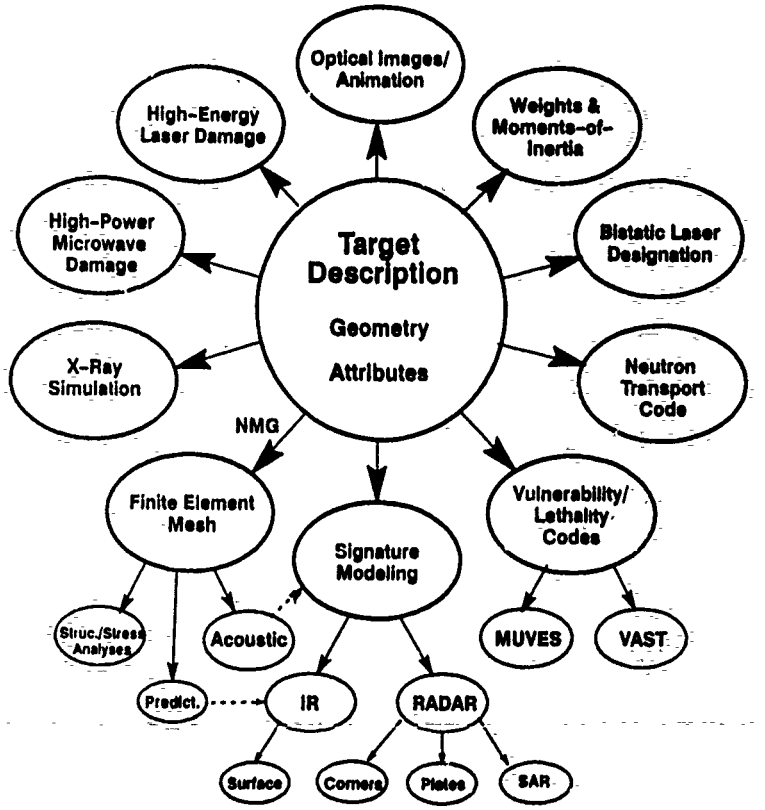


Figure A2. Current Applications of BRL-CAD.[1-7,10,15]

data base from time to time and the BRL-CAD package has a number of utilities written expressly to aid that process.

Second, specific interface standards have been adopted at the points where geometry is normally passed to applications (*e.g.* ray casting, topology, surface mesh). By this strategy changes which take place in the geometric data base itself are isolated from the specific means of geometry linkage. By means of this strategy both (a) an extremely large primary data base of geometry types can be supported in a data-storage efficient fashion and (b) various application codes can link to that data base in the manner most suitable for the application.

As noted above, although the original application of BRL-CAD was in support of vulnerability and lethality analyses, its utility has been extended much beyond those applications.[1-7,10,15] A graphical depiction of current uses is given in Fig. A2.

ACKNOWLEDGMENT

This work has been supported in part by the Joint Technical Coordinating Group-Munitions Effectiveness (JTCG-ME), Smart Munitions Working Group, under grants monitored by Julian A. Chernick, US Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, 21005-5066.

REFERENCES

1. Paul H. Deitz, *Solid Geometric Modeling - the Key to Improved Materiel Acquisition from Concept to Deployment,* **The Proceedings of the XXII Annual Meeting of the Army Operations Research Symposium**, 3-5 October 1983, Ft. Lee, VA, pp. 4-243 to 4-269; also **Ballistic Research Laboratory Memorandum Report BRL-MR-3383**, September 1984.

2. Paul H. Deitz, *Modern Computer-Aided Tools for High-Resolution Weapons System Engineering,* **Proceedings of the CAD/CAM Mini-Symposium, MTAG 84**, 28-29 November 1984, Seattle, WA, pp. 23-42 (Published by US Army Industrial Base Engineering Activity, Rock Island, IL); also **Ballistic Research Laboratory Memorandum Report BRL-MR-3605**, February 1988 (AD-A197855).

3. Paul H. Deitz, *Computer-Aided Techniques for Survivability/ Lethality Modeling,* **Proceedings of the ADPA Survivability and Vulnerability Symposium IX**, 28-30 October 1986, Naval Surface Weapons Center, White Oak, MD; also **Ballistic Research Laboratory Memorandum Report BRL-MR-3667**, February 1988 (AD-A197893).

4. Paul H. Deitz, *Predictive Signature Modeling via Solid Geometry at the BRL,* **Proceedings of the Sixth KRC Symposium on Ground Vehicle Signatures**, Houghton, MI, 21-22 August 1984; also **Ballistic Research Laboratory Memorandum Report BRL-MR-3664**, February 1988 (AD-A197983).

5. Paul H. Deitz, *Predicting Signatures of Military Targets,* **Signal Magazine**, Journal of the Armed Forces Communications and Electronics Association, **Vol. 43**, No. 5, January 1989, p. 43.

6. Joseph C. Faison *The Electromagnetic Code Consortium,* **IEEE Antennas and Propagation Society Magazine, Vol. 32**, No. 1, February 1990, pp. 19-23.

7. Michael J. Muuss, *Understanding the Preparation and Analysis of Solid Models,* **Techniques for Computer Graphics**, ed. Rogers and Earnshaw, Springer-Verlag, 1987.

8. *MAGI, A Geometric Description Technique Suitable for Computer Analysis of Both Nuclear and Conventional Vulnerability of Armored Military Vehicles,* **MAGI Report 6701, AD847576**, August 1967.

9. M. J. Muuss, P. C. Dykstra, K. A. Applin, G. S. Moss, P. R. Stay and C. M. Kennedy, *A Solid Modeling System and Ray-Tracing Benchmark Distribution Package,* **Ballistic Research Laboratory CAD Package, Release 3.0, SECAD/VLD Computing Consortium**, 2 October 1988.

10. P. H. Deitz, T. M. Muehl, S. L. Henry, G. S. Moss, E. O. Davisson and S. A. Coates, *Synthesized CAD Methods for Combat Vehicle Survivability Analysis,* **Proceedings of the TACOM/ADPA Combat Vehicle Survivability Symposium**, 26-29 March 1990, National Institute for Standards and Technology, Gaithersburg, MD; also **Ballistic Research Laboratory Memorandum Report**, In Press.

11. Charles L. Arnold, Jr., *SRIM User's Manual,* **United States Army Intelligence and Threat Analysis Center Final Report**, September 1987.

12. Jorge V. Geaga, *Synthetic Aperture Radar Target and Terrain Simulator,* **NRTC Technical Report 86-4R**, December 1985.

13. John Pelfer, *Georgia Tech Research Institute Radar Cross Section Modeling Software,* Modeling and Analysis Division, Georgia Tech Research Institute, October 1986.

14. J. W. Crispin, Jr. and A. L. Maffett, *Radar Cross-Section Estimation for Complex Shapes,* **Proceedings of the IEEE**, August 1965, pp. 972-982.

15. Paul H. Deitz, William H. Mermagen, Jr., and Paul R. Stay, *An Integrated Environment for Army, Navy and Air Force Target Description Support,* **Proceedings of the ADPA Tenth Annual Symposium on Survivability and Vulnerability**, 10-12 May 1988, San Diego, CA; also **Ballistic Research Laboratory Memorandum Report BRL-MR-3754**, May 1989.

16. Michael J. Muuss, *Workstations, Networking, Distributed Graphics and Parallel Processing*, **Computer Graphics Techniques: Theory and Practice**, ed. Rogers and Earnshaw, Springer-Verlag, 199(

17. R. F. Harrington, *Field Computation by Moment Meth·'s*, Krieger, Malabar, FL, 1982.

18. Kevin J. Weiler, *Edge-based Data Structures for Solid Modeling in Curved-Surface Environments*, **IEEE Computer Graphics and Applications, Vol. 5**, No. 1, January 1985, pp. 21-40.

19. Kevin J. Weiler, *The Radial Edge Structure: a Topological Representation for Non-Manifold Geometric Modeling*, **Geometric Modeling for CAD Applications**, edited by M. Wozny, H. McLaughlin and J. Encarnacao, Springer Verlag, December 1987.

20. Michael J. Muuss and Lee A. Butler, *Boundary Representations, Combinatorial Solid Geometry, and n-Manifold Geometry*, **Advanced Computer Graphics Techniques**, ed. Rogers and Earnshaw, Springer-Verlag, In Press.

21. H. Gouraud, *Continuous Shading of Curved Surfaces*, **IEEE Transactions on Computers, Vol. C-20**, No. 6, June 1971, pp. 623-628.

22. Gary Laguna, *Recent Advances in 3D Finite Difference Mesh Generation Using the BRL-CAD Package*, **BRL-CAD Symposium '89**, Aberdeen Proving Ground, MD, 24-25 October 1989, pp. 21-35.

23. B. E. Morey, S. R. Stewart and K. E. Gunderson, *Infrared Image Simulation of Threat Vehicles*, **The Proceedings of the Test Technology Symposium**, The Naval Surface Warfare Center, 18-20 April 1989.

24. William R. Reynolds, **PRISM User's Manual, Version 2.0**, Keweenaw Research Center, Michigan Technological University, Houghton, MI, October 1989.

25. Jack Jones, *BRL-CAD To Flat Facet Conversion*, **Proceedings of the BRL-CAD Symposium '89**, supported by the American Defense Preparedness Association, Aberdeen Proving Ground, MD, 24-25 October 1989.

26. Susan A. Coates and Edwin O. Davisson, *Conversion of a BRL-CAD model to a PRISM Model*, **Proceedings of the Conference on Ground Target Modeling and Validation**, Keweenaw Research Center, Houghton, MI, 21-23 August 1990; also **Ballistic Research Laboratory Memorandum Report**, In Press.

27. Gary S. Moss, *The "lgt" Lighting Model*, **BRL-CAD Symposium '88**, Aberdeen Proving Ground, MD, 28-29 June 1988, pg. 11.

28. P. Bezier, *Mathematical and Practical Possibilities of UNISURF*, Academic Press, New York, 1974.

29. Private communication with Kenneth Brand, Aircraft Division, Northrop Corporation.

30. John B. Schneider and Irene C. Peden, *Differential Cross Section of a Dielectric Ellipsoid by the T-Matrix Extended Boundary Condition Method*, **IEEE Transactions on Antennas and Propagation, Vol. 36**, No. 9, September 1988.

31. Irene C. Peden, John B. Schneider and John Brew, *Electromagnetic Detection of Buried Dielectric Targets*, **Proceedings of the 1990 International Geoscience and Remote Sensing Symposium, Vol. 1**, College Park, MD, May 1990, pp. 15-18.

THE BRL-CAD PACKAGE

VOLUME I
The BRL-CAD
Philosophy

RELEASE 4.0
1991